Modelling Tools and Methodologies for Rapid Protocell Prototyping

James Smaldon, BSc.

Thesis submitted to the University of Nottingham

for the degree of Doctor of Philosophy

July 2011

Abstract

The field of unconventional computing considers the possibility of implementing computational devices using novel paradigms and materials to produce computers which may be more efficient, adaptable and robust than their silicon based counterparts. The integration of computation into the realms of chemistry and biology will allow the embedding of engineered logic into living systems and could produce truly ubiquitous computing devices. Recently, advances in synthetic biology have resulted in the modification of microorganism genomes to create computational behaviour in living cells, so called "cellular computing". The cellular computing paradigm offers the possibility of intelligent bacterial agents which may respond and communicate with one another according to chemical signals received from the environment. However, the high levels of complexity when altering an organism which has been well adapted to certain environments over millions of years of evolution suggests an alternative approach in which chemical computational devices can be constructed completely from the bottom up, allowing the designer exquisite control and knowledge about the system being created. This thesis presents the development of a simulation and modelling framework to aid the study and design of bottom-up chemical computers, involving the encapsulation of computational reactions within vesicles. The new "vesicle computing" paradigm is investigated using a sophisticated multi-scale simulation framework, developed from mesoscale, macroscale and executable biology techniques.

Acknowledgements

Firstly, I would like to thank my supervisor Professor Natalio Krasnogor, for his immense enthusiasm, helpful advice, careful guidance and the interesting discussions we have had over the course of my PhD. I also want to acknowledge everyone in ASAP, in particular Jonathan Blakes, Francisco Romero-Campero and Paweł Widera for so many interesting discussions regarding life, the universe and programming. I would like to give thanks to my parents for their love and support and for giving me an interest in science and research from an early age. Most of all, I would like to thank Victoria Spurrett for her patience, love, understanding and support over the last four years, without which I could not have written this thesis. Finally, I would like to acknowledge the Engineering and Physical Sciences Research Council (EP/017215/1) and the school of computer science for the funding my PhD.

To my parents.

Contents

List of Figures

List of Tables

Nomenclature

2/3-SAT The 2/3 variable Boolean satisfiability problem

BRN Biological regulatory network

Chell Artificial chemical cell

CSL Continuous stochastic logic

CTMC Continuous time Markov chain

CUDA Compute Unified Device Architecture

DMPC Dimyristoylphosphatidylcholine

DNA Deoxyribonucleic Acid

DPD Dissipative Particle Dynamics

GFP Green Fluorescent Protein

GPU Graphics Processing Unit

GRN Gene regulatory network

LPP system Lattice Population P system

MD Molecular Dynamics

mRNA Messenger Ribonucleic Acid

ODE Ordinary Differential Equation

PE Phosphatidylethanolamine

PEG-PLA Polyethlyene Glycol-Polylactide

PEO-PEE Polyethylene Oxide-Polyethylene Ethylene

PEO-PLA Polyethylene Oxide-Polylactide

PRN Pseudo Random Number

PRNG Pseudo Random Number Generator

RNA Ribonucleic Acid

SM Streaming Multiprocessor

SP system Stochastic P system

SSA Stochastic Simulation Algorithm

UC Unconventional Computing

Chapter 1

Introduction

This thesis presents the development of a detailed multiscale simulation framework for the study of a proposed biomolecular computing paradigm *Vesicle Computing*. In this chapter the concept of vesicle computing is introduced and discussed in the context of unconventional computing, alongside the multiscale modelling approach to the study of this new computing paradigm.

1.1 Background and Motivation

Although the Turing machine model and Von Neumann architecture provide an excellent platform for theoretical reasoning about computation, they do not consider the properties or limitations imposed by the materials which a physical embodiment of a computer might be constructed from. The continued development of silicon transistor fabrication has maintained a steady increase in computational power over the last thirty years, such that silicon based electronics have become ubiquitous in everyday life. However, to maintain this ubiquity a huge amount of effort must go into the design, manufacture and maintainance of complicated electronic devices which may only last a few years before failing or becoming outdated. These devices are also fragile, lacking robustness in terms of the response to hardware damage with no innate ability for regeneration or self-maintainance.

Research in the field of unconventional computing considers radically different implementations and models of computation [45]. Whereas current computational hardware is limited to environments which are suitable for delicate electronics, unconventional computing considers the possibility of computing systems which could operate in vivo, or may self-assemble at molecular scales and are self-maintaining, repairing, and replicating. Biological and chemical systems have been an inspiration to the development of computational devices from the very beginnings of the study of computational theory, when Von Neumann first investigated the possibility of creating a self-replicating automata [191]. Indeed, one of the most impressive results in unconventional computing has been that of DNA computing proposed in [10], which introduced the possibility of performing computation by harnessing the information encoding and harnessing properties of deoxyribonucleic

acid (DNA) molecules.

As the ability to manipulate living organisms through sophisticated bioengineering techniques improves, there is an increasing motivation to explore the possibility of computing within the biological and chemical realms. By enabling a connection between existing electronic computing technologies and biochemical systems, it may become possible to interact with biological and chemical systems at the molecular level, enabling a control over biology which enables computing that is truly ubiquitous and embedded within the machinery of microbiological life. Existing computational devices can only provide the crudest of interfaces between electronic and biological systems, and so it will be necessary to consider how we can manipulate existing chemical and biological systems, or create entirely new ones, to create a new generation of pervasive computing devices, which will live, reproduce and maintain themselves with little or no outside interference.

Micro biological organisms such as bacteria have a number of properties which are desirable in an unconventional computing platform, such as their high energy efficiency, their ability to reproduce and adapt to different environments either over short time scales by the differential expression or repression of certain genes, or over longer timescales by the process of evolution. These features have inspired the concept of *cellular computing*. This new approach to biocomputing proposes to harness microbiological entities to perform computation by applying knowledge from the fields of systems and synthetic biology and augmenting the natural bacterial genome with new genes using the techniques that have been perfected in the field of genetic engineering over the last 30 years (Figure 1.1 shows a schematic of the cellular computing approach). Often, these implanted genes encode transcription factors which act as regulators of other genes via the mechanisms of transcriptional and translational regulation. By selecting genes which regulate one another, it is possible to produce networks of regulation that result in complex protein expression dynamics, such as positive and negative feedback which have been used to exhibit switch-like and oscillatory behaviours in E. *coli* bacteria. The ability to implant networks of transcriptional regulation within bacteria that produce switch-like behaviours should allow the designer of the cellular computing gene circuit to reason about the design in a more abstract way, by thinking in terms of modular abstractions such as logic gates, which can be created from regulatory networks and connected together to create the overall design, just as an electronic engineer can select from a set of standardised parts to create a circuit.

The intention in cellular computing is not that the computations occur more quickly relative to standard desktop computation (on the contrary, the gene network logic gates may take minutes or hours to stabilise after the input is changed) but is rather that the approach allows the "programming" of cellular devices which can interact with other bacteria and cellular organisms. Eventually, cellular computing may produce computational entities which reproduce to create colonies capable of large scale parallel processing, operating in environments which may be unsuitable for existing electronic computation. One obvious application is in medicine, where the development of computational entities which could operate within the body will enable programmed responses to disease

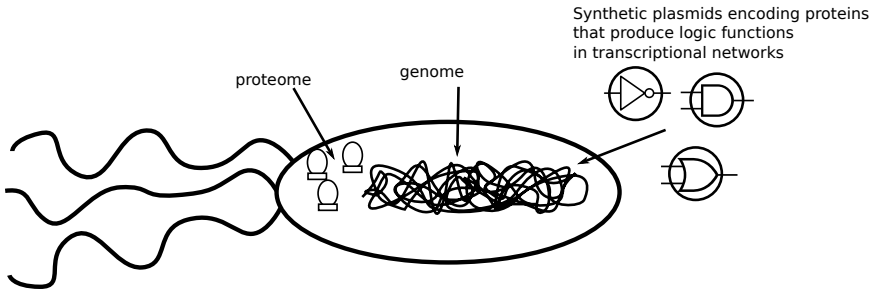


Figure 1.1: Cellular computing: An existing organism is augmented with synthetic networks of gene regulation, which can encode logic functions, by connecting these genes together (e.g. choosing proteins that regulate the expression of other genes, simple circuits can be constructed.)

indicators, enabling continuous disease monitoring and providing highly specific delivery of drugs. One can also imagine the use of cellular computers for the intelligent processing or digestion of waste materials to form new biofuels, or as chemical sensors which could clearly indicate the presence of toxins in foodstuffs for example.

Although the ability to implement simple logic gates as synthetic gene regulatory networks within living bacteria has been demonstrated in the lab, there has been considerable difficulty in scaling the modularised approach to gene regulatory network design in cellular computing to more complicated computational regulatory networks. These difficulties arise due to fundamental differences between the functioning of the regulatory networks in the cellular environment, and the electronic systems from which logic gates are derived. Namely, in an electronic circuit, the assumption is that components only interact with one another when there is a physical wire connecting them. In cellular computing, this assumption is no longer valid, as the components in the circuit are the gene regulation sites on the bacterial gene sequences, and all the transcription factors and other molecules inside the cell will be constantly moving within the cytoplasm due to the Brownian motion of the molecules. If an interaction is at all possible between genes and transcription factors, then it is therefore very likely to happen eventually, regardless of whether it was what the designer of the gene regulation network anticipated or intended. This means that it is difficult to minimise complexity and perform design in a modular fashion as the designer of a cellular computing circuit must be constantly aware of possible interactions between the module and all other systems in the cytoplasm.

This problem is compounded by the wide variation in the binding properties of the transcription factors and their target sites, and the very small numbers of molecules of each chemical species in the cytoplasm, leading to highly stochastic reaction dynamics. As well as this, when considering the reliability of cellular computing circuit design over longer timescales, modifying bacteria to compute in this way involves a conflict between the synthetic logic integrated into the bacterial genome, which will often force the cell to expend large amounts of energy expressing genes which are useful as part of the computation, but provide little benefit to the cell, and so evolutionary

pressures will result in the removal of those synthetic genes and systems which are a detriment to the organisms survival as soon as the organism is taken from a nutrient rich environment [259].

The difficulties presented by the cellular computing approach are then twofold: Firstly, there is the necessity of creating a set of well-characterised, reliable, composable and predictable components from networks of gene regulations which can be used to generate the simple logic functions required to form the basis of cellular computing, thereby allowing the designer of cellular computing systems to construct the system at a higher level of abstraction and secondly, the necessity of minimising the interactions between the bacterium's underlying metabolic, reproductive and autopoietic systems and the newly introduced transcriptional regulation systems. Work in the field of synthetic biology may provide inspiration in attempting to overcome these difficulties.

Synthetic biology marks a shift in the focus of investigation of living organisms from the study of the systems and networks of gene regulation that exist within natural bacterial genomes, to the engineering and modification of those genomes with synthetic gene circuits which may come from many different sources. With regards to the second challenge, that of minimising the interactions between the synthetic systems and the underlying systems of the bacterium, synthetic biology proposes the creation of a minimal life form, which would in turn reduce the probability of unexpected interactions. Two approaches to the creation of this minimal organism have been proposed which can broadly be categorised as "top-down" and "bottom-up". Both of these approaches intend to create a minimal organism, which has a genome containing genes encoding only those systems required for the organism's survival, but the two approaches attempt to arrive at this result from two opposite starting points (Figure 1.2 illustrates these two approaches). In the top-down approach, a species of bacteria is taken and the genome is reduced by a trial and error process of gene-knockouts to determine exactly which genes are indispensable, the end result is an organism with a minimal genome, that can be used as a synthetic biology "chassis" for the addition of synthetic gene systems. The bottom-up approach is complementary to the top-down approach, and is a more ambitious endeavour, involving the creation and combination of the essential systems for life entirely from scratch, using any chemical or biological system which might be useful. This minimal cell, created entirely from scratch, would then act as the foundation for the addition of new synthetic systems and functionalities.

Although much consideration has been given as to how computation might be embedded within existing bacteria, as part of the top-down approach, the possibility of creating living chemical computational devices entirely from scratch, following the bottom-up approach has not been considered[1]. This thesis presents the investigation of a route to the construction of living computers which is alternative and complementary to that of cellular computing: The creation of chemical computational entities from the bottom up. This approach combines the utilisation of useful molecular components from biology with the construction of new molecular machinery which will deliver

[1]Note that DNA computing does not count here, as a DNA molecule removed from its cellular context is not generally considered to be alive.

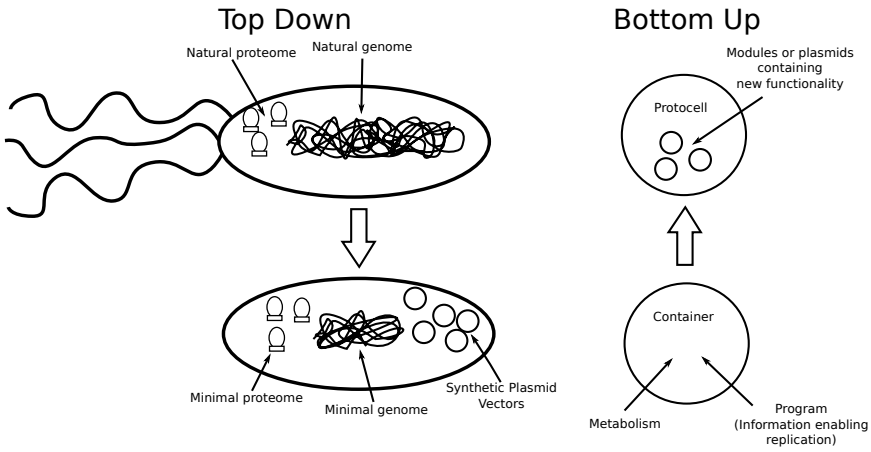


Figure 1.2: The top-down (left) and bottom-up (right) approaches to create a minimal organism in synthetic biology. In the top-down approach, the genome of an existing microorganism is reduced to a minimal set of genes required to sustain life, which can then be used as a platform upon which synthetic networks of gene regulation, encoding the desired functionality can be added and expressed within the cell. The bottom-up approach considers the creation of a minimal life-form entirely from scratch, by combining a protocellular container, metabolism and heritable information encoding. Additional functionality can then be added to the minimal cell as required.

precisely the functionality required to produce chemical computation. By designing everything from scratch, it should be possible to sidestep many of the inherent complications which occur when attempting to create computational devices by modifying existing organism from the top down, such as the likelihood of unintended interactions with the programmed system and the underlying bacterial machinery.

A key aspect of this approach is the encapsulation of the computational machinery within a self-assembled compartment, known as a vesicle, the boundary of which is demarcated by a fluid bilayer membrane which initially will be composed purely of amphiphiles. As well as providing simple chemical containers which can be formed inexpensively from self-assembly processes or by the application of simple lab procedures, vesicle membranes have advantages when compared with the application of existing microorganisms for computation. For example, evolution in plants and animals has evolved highly tuned immune responses to pathogenic microorganism membranes, meaning that any microorganism based medical cellular computing agent would be quickly targeted and destroyed. In contrast, vesicles can be formed from new materials such as copolymers, which remain undetected by the immune system. Since vesicles act as the containers for the proposed bottom-up cellular computer, the newly proposed approach is named “Vesicle Computing”.

The manner in which a vesicle computer will be constructed will involve the rational design of the implementations of the three key components (Container, metabolism and inheritable information) which will make up the protocell [220]. As every aspect of a bottom-up computational vesicle will be constructed from scratch, a design and engineering approach will be more appropriate than the reductionist approach to understanding existing systems which has been pursued in tradi-

tional and top-down synthetic biology. As well as proposing the vesicle computing paradigm, this thesis presents both a coherent methodology for the design of bottom-up protocell systems focussing specifically on computational chemical automatons, and a proposed simulation and modelling framework which enables the exploration of those designs in-silico. Although simulation and modelling techniques are unlikely to be a replacement for a laboratory implementation in the near future, the ability to explore the behaviour of possible protocell designs will allow the designer to gain a better understanding of the unintended consequences and behaviours of the design, and ease the creation of new protocell designs in the same way that computer aided design packages aid design in traditional engineering. The design techniques presented in this thesis are therefore intended to be used as part of a feedback cycle, involving the formal specification of a computational protocell design, which can then be simulated at a number of different scales, indicating flaws in the design which can then be modified and resimulated until it is finally validated in-vitro. The simulation should aid the vesicle computing endeavour by enabling the rapid prototyping of protocell models and thus investigating hypothetical vesicle computers. In turn, this will help guide experimental exploration and could contribute towards the reduction of the number of laboratory experiments required during the vesicle computing design cycle. As the understanding of the materials and processes required to construct a protocell improves, and increasingly powerful computational resources permit more detailed and accurate simulations, it is the long term goal that more and more of the protocell design work could be performed in-silico.

Simulation and modelling techniques are increasingly employed in synthetic biology, where the aim is the creation of well characterised and modular genetic components which can be used in the construction of complex biological systems. However, there are few simulation and modelling approaches that take into account the fact that biological systems work over different length and time scales. Understanding a system as being composed of units which in turn are composed of other smaller units is the essence of hierarchical reductionism. It is important to realise however that the abstractions of scale are constructs designed to aid understanding. All processes in a biological entity are emergent from interactions at the lowest possible level. Therefore, the creation of realistic high level design techniques for vesicle computing systems will only succeed if the assumptions and abstractions made at the high level are correct in terms of the low level interactions.

This thesis therefore makes the following key contributions to the development of a bottom-up approach to cellular computing:

- the proposal of a new approach to cellular computing, which follows the bottom-up approach in synthetic biology rather than cellular computing's top-down methodology. This new implementational approach is named "vesicle computing".
- A study regarding how simulation and modelling techniques might be applied to the development of both cellular and vesicle computing systems.

1.2 Research Questions

The research questions considered in this thesis are the following:

- How might a simple chemical computational device, encapsulated within a vesicle be designed?
- How can computer science aid the development of vesicle computing with simulation and modelling techniques?

1.3 Research Methodology

In this thesis a number of different research methodologies have been applied to the consideration of these questions. These can be summarised as follows:

1. A thorough background survey of the literature, with ongoing updates to this survey throughout the course of the research.
2. Computational experimentation.
3. Analysis of computational results.
4. Theoretical Analysis.
5. Links to wet lab experiment.
6. Software engineering.

The application of simulation and modelling, in the form of a newly developed framework for the study of vesicle computing has been at the core of the work undertaken, and the development of a vesicle computing simulation and modelling framework is described in full in chapter 3. In creating this framework, a more theoretical approach was taken when attempting to understand the relationship between the chemical reaction rate parameters in dissipative particle dynamics (DPD) and stochastic P systems. And an experimental approach was taken in studying some of the key aspects of the proposed vesicle computing paradigm in multi-scale simulation using the framework.

1.4 Contributions

This work focuses on the development of techniques and models for simulation and study of the proposed vesicle computing paradigm and synthetic biology systems. A multi-scale simulation framework using several different techniques, each suitable for particular length and time scales was designed and employed for this purpose. A number of contributions were made as a result of the work described in this thesis, and these are listed below

- **The Proposal of Vesicle Computing**
  This thesis presents the concept of vesicle computing and illustrates the concept with simulating and modelling techniques.

- **The design and creation of a new multi-scale modelling framework**
  A multi-scale simulation and modelling framework was developed for the investigation of vesicle computing systems. The framework enables the investigation of single vesicles in high detail all the way to entire populations of vesicles. The modelling framework includes a fast parallel implementation of the Dissipative Particle Dynamics method.

- **Fast parallel MPI and Nvidia CUDA based implementations of Dissipative Particle Dynamics**
  The development of the Dissipative Particle Dynamics implementation of this code involved the development of novel algorithms to enable the implementation of the method for the compute unified device architecture (CUDA) platform.

- **Study of vesicle self-assembly** As part of the investigation into the use of different amphiphiles for vesicle computing, a large scale investigation into the process of vesicle self-assembly was performed using the DPD software.

- **Methods for the characterisation of chemical reactions in DPD, enabling the conversion of parameters for use with other simulation techniques.** A chemical reaction scheme was added to the implementation of DPD, and the theoretical links between the chemical reaction rate parameters in dissipative particle dynamics and the stochastic simulation algorithm were investigated.

- **Investigation through simulation and modelling of communication in vesicle computing systems.** Transport of material across the vesicle membrane was investigated through highly detailed simulations of vesicles which contain two different kinds of pores. The diffusion rate of the vesicles was characterised.

- **Proposal and investigation through simulation of vesicle computers composed of encapsulated logic gates.** Chemical logic gates inspired by the functionality of gene regulation systems in bacteria were placed within vesicles and simulated. The dynamics of the reactions inside and outside of the vesicle were investigated in detail.

- **The application of model checking for protocell/chell model design** Model checking techniques were applied to the development of model chemical logic gates in vesicles. To the best of my knowledge, this is the first time that the application of model checking has been applied to bottom up synthetic biology systems in the literature.

1.5 Publications

During the work undertaken for this thesis, two journal papers, two peer reviewed conference publications, one book chapter and a full software release were produced. The details of these publications are as follows:

- James Smaldon, Jonathan Blakes, Natalio Krasnogor, and Doron Lancet. A multi-scaled approach to artificial life simulation with p systems and dissipative particle dynamics. In *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 249–256, New York, NY, USA, 2008. ACM. (Nominated for best paper award).
- James Smaldon, Natalio Krasnogor, Cameron Alexander, and Marian Gheorghe. Liposome logic. In *GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 161–168, New York, NY, USA, 2009. ACM.
- James Smaldon, Francisco Romero-Campero, Jonathan Blakes, Jamie Twycross, and Natalio Krasnogor. A cellular computing based solver for instances of the 2-SAT problem. *Journal of Logic and Computation*, Submitted, 2009.
- James Smaldon, Francisco J. Romero-Campero, Marian Gheorghe, Cameron Alexander, and Natalio Krasnogor. A computational study of liposome logic - towards cellular computing from the bottom up. *Systems and Synthetic Biology*, Volume 4, Number 3, pages 157–179.
- Lin Li, Peter Seiepmann, James Smaldon, German Terrazas, and Natalio Krasnogor. *Studies in Multidisciplinarity Volume 5: Systems Self-Assembly: Multidisciplinary Snapshots*, chapter Automated Self-Assembling Programming, pages 281–303. Elsevier, 2008.
- `http://www.infobiotics.org/DPD` Release of the DPD software and documentation

1.6 Structure of the Thesis

In the next chapter, a review of the literature is presented with the aim of placing vesicle computing in the context of non-conventional computing and synthetic biology research; published experimental results which have relevance to the study detailed in the rest of the thesis are examined and consideration is also given to the current state of the art in simulation and modelling techniques for synthetic biology. In Chapter 3 the framework used to perform the investigations of vesicle computing which are detailed in the rest of the thesis is presented. Chapter 4 focuses on the development of a key component of the simulation framework, a fast parallel implementation of the Dissipative Particle Dynamics with reaction extensions. In Chapter 5 the simulation, collection and analysis of a library of vesicle computational building blocks, inspired by the MIT registry of biological parts is described in detail and results of experimental exploration of the vesicle computing paradigm using the simulation framework are then reported. Chapter 6 contains a detailed investigation into the

rate of diffusion across porated vesicle membranes. Chapter 7 presents simulations of model vesicle computers based on networks of gene expression, and Chapter 8 illustrates the dynamics in simulation of a population of vesicle computers. The thesis is concluded in Chapter 9 with a discussion of the simulation and modelling techniques used in the context of the design of a vesicle computer, and possible routes for a chemical implementation of the paradigm, and the direction of future research are considered.

Chapter 2

An Overview of Unconventional Computing

In this chapter, the context in which vesicle computing is developed is elucidated by considering the historical and theoretical development of the conventional computing paradigm, and then unconventional computing approaches are discussed in combination with a review of the literature. As vesicle computing is inspired by attempts to embed computation within living organisms in synthetic biology, the current cellular computing research in the context of synthetic biology is considered and the literature in this area is reviewed. Special attention is paid to existing work which relates to the so called "bottom-up" approach to a minimal cell in synthetic biology, and the application of simulation and modelling techniques in synthetic biology.

2.1 Conventional Computing

John Von Neumann first proposed the concept of a stored program machine architecture, where a machine stores the processing instructions within the same memory address space as the data to be processed [260]. This was a major breakthrough in computer implementation, as it enabled the execution of programs that could modify themselves. This architecture represented the first implementable approximation of a universal Turing machine, able to simulate the computation performed by any other Turing machine program-input pairs (Figure 2.1 shows a schematic representation of this architecture).

The development of the transistor in 1947 by John Bardeen, Walter Brattain and William Shockley, enabled the process of miniaturisation in electronics, and large numbers of transistors could be placed within a single package in the form of a microchip [19]. As the demand for computing power increased, so did the number of transistors contained within a typical processor, whilst the size of transistors rapidly decreased. Perhaps due to the rapid progress which has been made in silicon based processor manufacture and design, the implementation of the computing paradigm against which modern processors are developed has not been altered greatly from that which was formulated by Turing, Von Neumann and others.

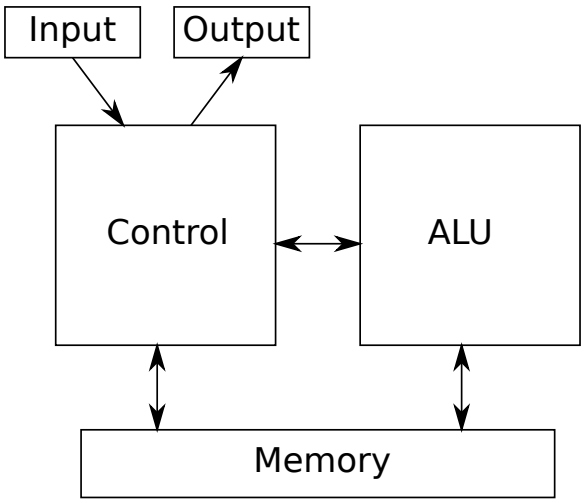


Figure 2.1: The Von Neumann architecture, composed of a control unit, an arithmetic logic unit (ALU) and a memory, which holds both instructions and data.

2.1.1 The Theory of Computing and the Standard Paradigm

Mathematician D Hilbert first posed the question "is it possible to find an algorithm for determining the truth or falsehood of any mathematical proposition?". The study of this question and other similar questions led Alan Turing to create an abstract mathematical representation of a computational device, the Turing machine, which consists of a *head*, which can be in any of a finite set of states, and a *tape* which is infinitely long and is divided into cells, each holding one of a finite number of symbols. Depending on the state which the head is in and the symbol of the cell which the head is reading, the head will change state, write a symbol to the current cell, and then move one cell to the left or right. To perform a computation, the input to the machine is written to the tape, and the tape head is positioned on the start symbol, the head then reads the first symbol and changes state and moves according to the symbols and the state transition table for the head, until a special symbol, which is a member of the set of halt states is reached, at which point the computation finishes [130]. The Turing machine can be defined formally by a 7-tuple:

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F) \tag{2.1}$$

where:

Q: is the finite set of states which the head can be in.

Σ: The finite set of input symbols.

Γ: The finite set of tape symbols; Σ is always a subset of Γ.

δ: The transition function. The arguments of $\delta(q, X)$ are a state q and a tape symbol X. The value of $\delta(q, X)$, if it is defined in a tuple (p, Y, D) where:

p: is the next state, in Q.

Y: is the symbol, in Γ, written in the cell being scanned, replacing whatever symbol was there.

D: is a direction, either L or R, which indicates the direction in which the head moves.

q_0: The start state, a member of Q, in which the head is placed initially.

B: The blank symbol, which is in Γ but not in Σ. The blank symbol is placed initially on all cells of the tape where the input symbols are not placed.

F: The set of final or accepting states, a subset of Q.

An important consequence of this formalism is that the Turing machine may not necessarily halt for a given input, as the head can move left and right along the tape, which permits the expression of infinite loops. The problem of determining whether a Turing machine will halt for a given input is known as the halting problem, and in 1936 Turing showed that a general algorithm for solving the halting problem for all possible program-input pairs cannot exist. The halting problem was one of the first problems which was proved to be undecidable and showed that there are certain programs which cannot be computed. Also in 1936, Church hypothesised that any computable program can be expressed in the Lambda calculus, a functional notation that is equivalent in terms of computational power to a Turing machine [58]. This led to the Church-Turing Thesis, which hypothesises that all models of computation will be at most equivalent in power to a Turing machine. These results, together with Gödel's indicate that the answer to Hilbert's question is no, in that for an arithmetical system to be consistent (i.e. not permitting logical contradictions), it must be incomplete (i.e. unable to compute every function).

Although Turing's results consider the limits of what can be computed, they say nothing about the resources required to perform a computation in terms of the size of the tape or the number of moves the head of the Turing machine must make. In order to mechanically compute a solution to a given problem, the problem must not only be decidable but should also be feasible in terms of the amount of time required for the computation to complete. The measure of the amount of resources in terms of the size of the tape and the number of head movements required to compute a solution for a given program/input pair is known as the program *complexity*. The field of computational complexity focuses on classifying computational problems according to this metric.

The NTM can be used to define another class of problems, the non-deterministic polynomial or *NP* problems which can be solved in polynomial time with a NTM but not with a standard Turing machine. Clearly, this set of problems includes all of the problems in P, as an NTM could execute a program with P complexity in polynomial time without ever needing to branch. The *NP-complete* class is a particularly interesting subclass of the NP problems, as all problems in NP can be reduced (converted) to the NP-complete problems within polynomial time. Those problems that are not in P but in NP are decidable, but are intractable to solve even for relatively small instances.

NAND Gate			NOR Gate		
In X	In Y	Out	In X	In Y	Out
T	T	F	T	T	F
T	F	T	T	F	F
F	T	T	F	T	F
F	F	T	F	F	T

Table 2.1: Truth tables for the logic gates NAND and NOR.

2.2 Unconventional Computation

Unconventional computing is a research area exploring methods of computation which may differ from the standard computational paradigms [7, 5, 9]. Typically, unconventional computational devices fall roughly into three categories, those which provide more or less computational power than a Turing machine (hyperturing or hypoturing computation), those which may be Turing equivalent, but are not Von Neumann equivalent in terms of their implementation, and those which may be Turing and Von Neumann equivalent, but implemented using novel materials which may permit program execution in exotic environments. By exploiting novel materials and systems for computation, it may be possible to exceed the standard model in terms of computational power and efficiency [238].

Many unconventional computing (UC) approaches focus on the implementation of logic gates in new media or environments. Logic gates are the physical implementations of Boolean logic functions, which produce a Boolean output based on the values of one or two Boolean inputs. Some logic gates are known as "universal" in that they can be connected together to reproduce the function of any other logic gate, and hence produce a computational device which is Turing universal. Examples of universal gates are Not-And (NAND) and Not-Or (NOR). Figure 2.1 shows the truth tables for these logic gates.

There are a number of examples of chemical implementation of logic gates. For example, Adamatzky and De Lacy Costello proposed an implementation of "Wave computation" in which an exclusive or (XOR) logic gate was constructed from a gel containing palladium chloride [4], which was cut into a T shape. Inputs to the logic gate were encoded as the presence or absence of potassium iodide at the two endpoints at the top of the "T". If the input reactant was placed on one of the arms of the T, then a reaction occurred between the palladium chloride and the potassium iodide, resulting in a colour change in the area of the gel where the reaction occurred. As the reactant diffuses across the gel, a wave of reaction propagates to the bottom of the T shape, which corresponds to the logic gate outputting a "high" signal. If the reactant was placed on both input arms of the T shaped gel simultaneously, then the reaction waves from both sides meet in the centre of the T and "collide", and the wave does not propagate any further, which corresponds to the logic get outputting a "low" signal. Figure 2.2 shows the action of the logic gate in schematic form. The authors also illustrated

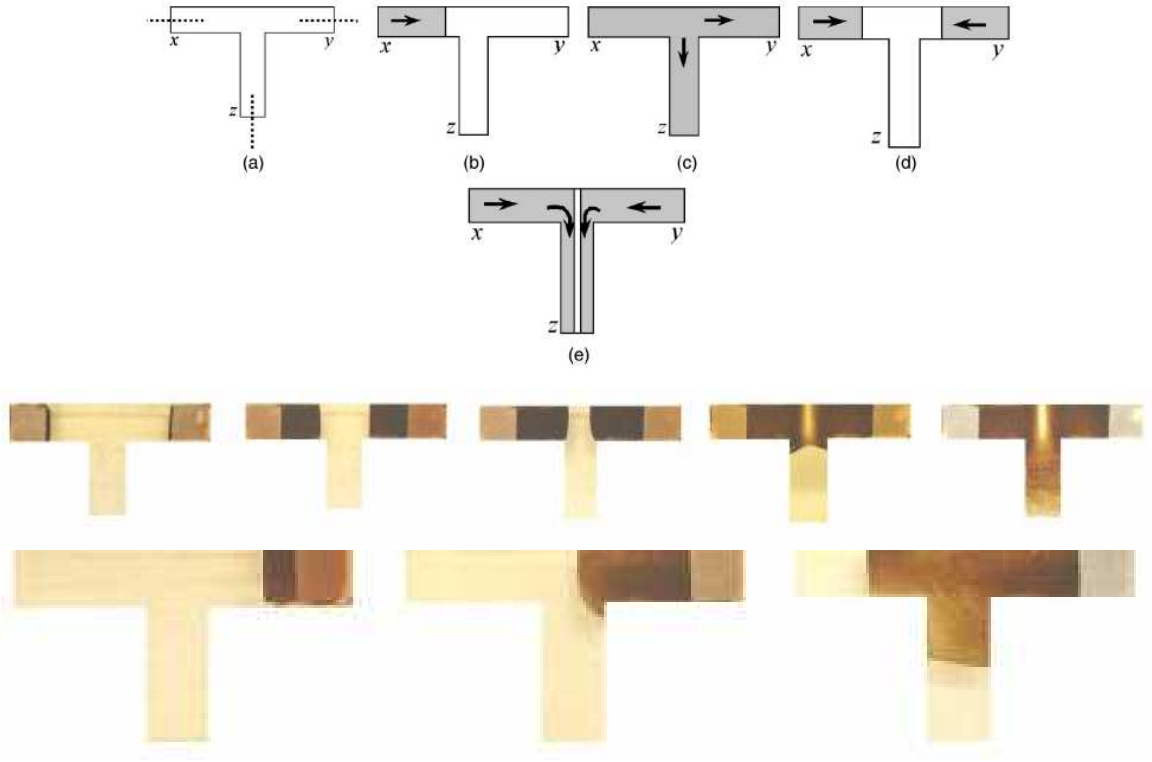


Figure 2.2: The diagram (Top) shows a schematic representation of wave computation. The logic gate is composed of a T shaped reactive medium. Inputs to the gate are set by the absence or presence of reactant on the arms of the T shape. When a single input is present (b) the reactant diffuses across the entire gate and reaches the output at the bottom of the T shape (c). When both inputs are present (d) the reactants interact and the result is that the reactant is prevented from reaching the output (e). The middle row of images shows the result of the chemical reactions when both inputs were present, and the bottom row of images shows the result of the chemical reactions when only the right hand input was present. Reprinted Figures with permission from [4]. Copyright 2002 by the American Physical Society.

the implementation of more complex three input logic gates constructed from gel cut into more complicated shapes.

INHIBIT logic gates implemented as polymers were proposed in [115]: Two different polymer logic gates were created, in the first the input to the system was the pH of the system, and in the second the temperature was the input. In [187] the authors showed the coupling of an information processing enzyme based system with pH responsive materials, the result of which was reversible changes to the assembled structure of the nano particles based on the logical inputs to the system. Magri proposed a fluorescent AND logic gate which takes as input proteins and electrons [172].

The possibility of using droplets and bubbles in micro fluidics for computation was proposed by Prakash and Gershenfeld [216] and was reviewed by Epstein [82]. The micro fluidic logic gates function due to a property of bubbles travelling within microscopic channels, which means that if a

bubble approaches a fork in the channel, it will take the path of least resistance. However, if a bubble has recently travelled down a path then the resistance increases. By moderating the frequency at which bubbles travel down the paths, complex sequences of path choices could be made.

One promising unconventional computation paradigm is quantum computing, first proposed as a theoretical possibility in [70], and later explored by Loss and DiVincenzo [165]. Rather than representing data as bits, as in conventional computing, the data is represented as qubits (quantum bits). Qubits differ from conventional bits in the way that a number can be represented. As well as the standard representations, each qubit can be in a superposition state, in which it may be in one or another of the other states. It should be noted that a quantum computer does not exceed a Turing machine in terms of the functions which it can computer, but instead allows the computation to be performed more efficiently, by exploiting quantum parallelism. A method by which this parallelism could be exploited was proposed in [237], who demonstrated a polynomial time solution to the problems of integer factorisation and the finding of discrete logarithms.

Light is proposed as a medium for performing computation in an unconventional computing approach known as "photonic computing" or "optical computing". In optical computing, electrons within a processor are replaced with photons, and since the light releases much smaller amount of heat energy in comparison with an electrical current within a wire, this may allow faster processing speeds and smaller devices [227]. Another optical computing approach involves delaying light signals to encode solutions to NP-complete problems, which produces faster computation of solutions at the cost of an exponential increase in energy use in relation to the problem size [202].

Several authors investigated whether simple chemical reactions could be used for computation. In Hjelmfelt et al. The authors provide analytical results from a clocked neural network composed of "chemical neurons" which has been shown to be Turing universal [127, 126, 128]. In [171], the author shows that chemical kinetics can be Turing universal, and focuses on a key problem which occurs in the implementation of a chemical computer, in that the interconnections between chemical computing components result in signal loss as the signal is propagated deeper into the network. The author used dynamical systems theory to show that in order to produce a simple signal repeater from chemical reactions, reactions with double stoichiometry were necessary. The author then showed the creation of 4 logic gates and constructed a simple adder circuit.

2.2.1 Biological Computing

One prominent theme within unconventional computing is the proposal to make use of the information processing capabilities of biological systems. Biological systems have a number of desirable features such as robustness, adaptability, self-maintainability, self-reproduction and self-correction for example. The information processing behaviours of living organisms have been an inspiration for the development of novel computational paradigms. The understanding of the encoding of protein structures within sequence of DNA supports the interpretation of living cells as biological computing

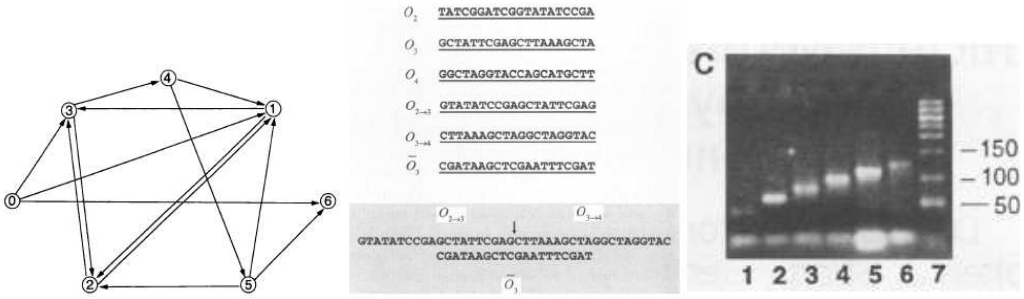
devices, processing information stored as DNA to maintain the machinery necessary for the continued functioning of the cell [137, 158, 64]. Coupled with the desirable properties of bacterial systems mentioned previously this means that biology presents a rich resource for the inspiration of new UC paradigms, and a number of different designs have been proposed, some focussing on utilising DNA alone for computation, and others attempting to harness entire cells for programmable computation.

The key application of biological computing systems will likely be the addition of controlled computation to biological environments, enabling fine grained control and manipulation of existing biological systems ranging from management of microbiological colonies to smart drug delivery and medical monitoring in humans. Other possible applications of biological and chemical computing might involve the development of in materio computing, the addition of computational functionalities to existing materials and surfaces. These materials could sense and react to changes in the environment and form the basis of truly pervasive computing. Furthermore biological, cellular and chemical computing could be utilised to provide a reliable and programmable interface between silicon based technology and the biological and chemical realms. Some concrete examples of biologically inspired unconventional computing are now considered.

Biomolecular Computing

There is an enticing link between the DNA code, which stores information in a sequence of nucleotides, and a stored program. Just as a set of algorithms specified as a computer program can be executed to produce behaviours which are vastly more complicated than their specification, the DNA sequence is a static encoding of protein sequences, which in turn interact to produce all of the signal processing and information processing behaviours required for life. The investigation into the possibility of using DNA for computation was one of the earliest successes in biomolecular computing.

Adleman was the first to show that DNA could be used for computation in this way [10], in an approach that harnessed both the specificity of the DNA sequence base pair bonding, and the massively parallel nature of chemical interactions in solution. The Adleman experiment showed the solution of the Hamiltonian path problem by encoding instances of the problem as strands of DNA. The Hamiltonian path problem is NP complete, and can be stated as follows: Given a graph with a set of nodes n and a set of edges e determine whether the graph contains a path p such that all element of n are contained in p and each element of n is only contained within p once. The Hamiltonian path is then the path through the graph which visits every node exactly once. Adleman solved the problem for a graph containing 7 nodes by numbering the nodes and encoding using three nucleotides the number for a node within a DNA sequence, Figure 2.3 illustrates the solution of the Hamiltonian path problem using DNA. The sequence for a candidate solution to the problem was 21 nucleotides in length. The computation of the solution proceeded by a process of generating



(a) (b) (c)

Figure 2.3: DNA Computing, images from [10]. Reprinted with permission from AAAS. (a) shows a directed graph. There is a hamiltonian path between nodes 0 to 6 as follows: $0 \rightarrow 1{,}1 \rightarrow 2{,}2 \rightarrow 3{,}3 \rightarrow 4{,}4 \rightarrow 5{,}5 \rightarrow 6$. (b) shows how a graph is encoded into DNA. Each solution is encoded by a random oligonucleotide of 21 bases and the images shows three such encodings (O_1,O_2 and O_3). An edge joining vertex i and j in the graph is represented by a 20-mer oligonucleotide, which contains a sequence which is complementary to half of i's sequence and half of j's sequence. (c) shows the gel electrophoresis for the result of the computation in which the Hamiltonian path is shown in lanes 1-6 (lane 7 is the molecular weight marker).

randomly every possible solution to the problem, by mixing free nucleotides in solution, such that the nucleotides would collide and bind randomly, to form random sequences. The algorithm proceeds via a number of sequential screening processes. The first involves removing all sequences which are not of the right length, so that only the DNA sequences containing 7 nodes remain. The next process involves removing all those candidate solutions which contain a node more than once, and then all sequences which encode solutions where the nodes are not adjacent. At the end of these screening processes, the only solutions that remain must be solutions to the Hamiltonian path problem, and so those strands of DNA can be extracted and sequenced to determine what the correct solution is. In this case, the benefit of DNA computing is that the massively parallel processing nature of molecular interactions and chemical reactions can be harnessed to compute the solution of NP complete problems more quickly than traditional computational devices.

Although Adleman's experiment was important in proving that the concept of DNA computing was feasible, there are a number of problems with this kind of approach to computation using DNA. Firstly, the computation took several weeks to complete, with each stage requiring the extraction and purification of the next set of candidate sequences from solution. The DNA computation was also not a general solution, solving only the 7 node instance of the Hamiltonian path problem. Although larger instances could be encoded with a longer DNA representation for each node, it is not clear whether errors in DNA sequence binding etc may limit the size of possible solutions. Gehani and Reif proposed a method of automating Adleman style DNA computation with the aim of overcoming some of the difficulties of the approach [101] in the form of micro-flow biomolecular computation, where the separate stages of the DNA computation could be performed in parallel in a

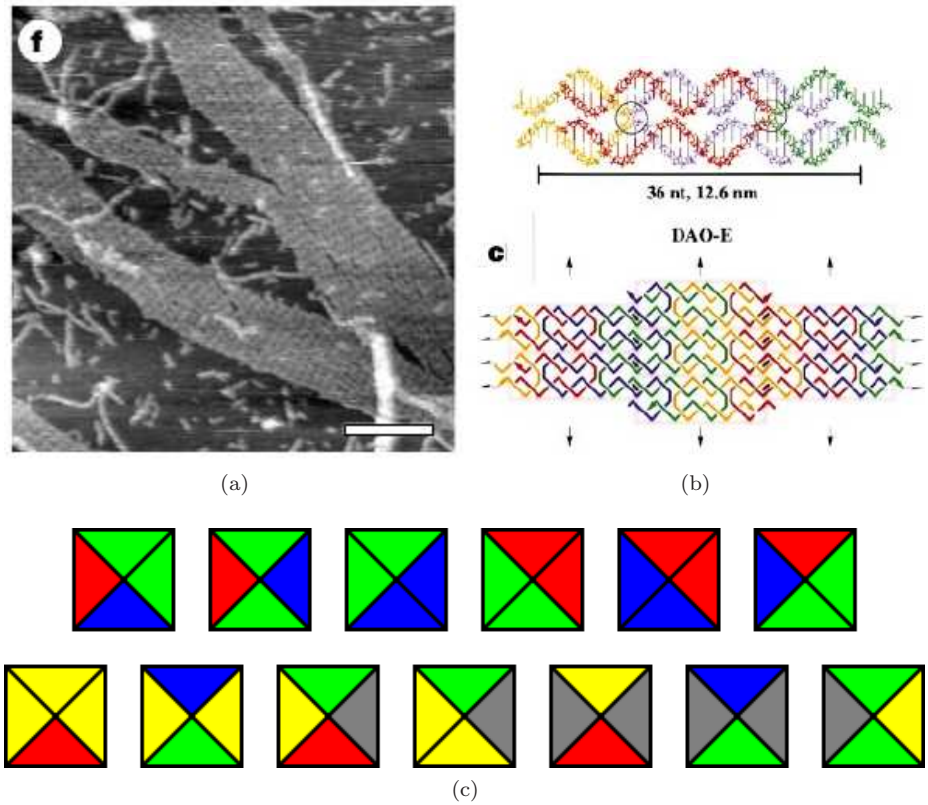
micro-electrical mechanical system (MEMS), with flow of DNA between the different compartments.

Adleman's approach was generalised by Lipton [162] who demonstrated the application of the generalised approach to the solution of the Boolean satisfiability (SAT) problem. The computational power of DNA computing was considered by Boneh et al. [39], who proved that DNA-based computers can be used to solve the satisfiability problem for Boolean circuits. Braich et al. showed that a similar approach could be used to create a DNA computation capable of solving a 20 variable instance of the 3-SAT problem [41]. A programmable DNA computer was first introduced by Benenson et al. [33], who created a simple automata from DNA sequences and DNA manipulating enzymes. The Benenson automaton acts by encoding the program as a sequence of instructions and a set of state transition sequences, the executed instruction is then removed from the DNA sequence and the computation proceeds by repeating the process for the next instruction.

The DNA automaton concept was extended by Benenson et al. [32] who created an automaton that was able to sense messenger ribonucleic acid (mRNA) levels in a living cell, which indicated the presence of prostate cancer. If no mRNA was present, then the DNA program was not executed. However, if the mRNA was present and bound to the automaton, then an mRNA sequence was produced which acted as a drug. A design for a DNA Turing machine is proposed by Shapiro et al. [229], who argue that in principle there is no reason why a fully programmable Turing machine could not be implemented with DNA. Stojanovic et al. [249] illustrated a DNA based automaton, which is capable of playing a game of tic-tac-toe interactively with a human opponent.

Another approach to performing computation with DNA molecules is the work of Erik Winfree, in which DNA helices linked together at crossover junctions are used to form "DNA tiles" [268, 269]. Sticky ends at the edge of each tile enable the tiles to self-assemble into regular lattice structures. By creating tiles from different DNA sequences, it becomes possible to control the self-assembly of the lattice to form patterns, and this approach has been shown to be able to create circuit patterns which could be used in a modular fashion to perform computation [62]. Wang tiles provide a theoretical model of this behaviour, which has been used to show that tile self-assembly can be used to perform computation, and is in fact Turing complete [245]. The automated programming of systems of Wang tiles was investigated by Terrazas et al. who applied evolutionary algorithms to the design of the tile surfaces with specific patterns [255, 156]. Figure 2.4 illustrates the structure of the DNA lattice and Wang tiles.

In the next section, this review focuses on a specific new unconventional computing paradigm, *cell computing* which is the inspiration for the vesicle computing paradigm proposed in this thesis, and considers the possibility of performing computation in living microorganisms by manipulating networks of gene regulation. This paradigm is at the forefront of a novel endeavour in biology and chemical systems research, the quest to produce *Synthetic Biology* systems.



(a) (b)

(c)

Figure 2.4: (a) shows an electron microscope of a DNA tile lattice, and (b) shows an example of the structure of the DNA in each tile (Figure adapted by permission from Macmillan Publishers Ltd: Nature [269], copyright 1998), (c) shows a selection of Wang tiles. Tiles will only stick together where the colours of the two sides are the same.

2.3 Top Down Synthetic Biology and Cellular Computing

Advances in the field of genetic engineering and biology allow increasingly precise control over the functioning of the networks of gene regulation in bacteria. Whereas previously the focus was on understanding the function and expression of individual genes, systems biology views biological systems at a higher level of abstraction, attempting to ascertain how the networks of transcriptional and translational regulation that occur in the bacterial genome can be analysed and understood. Recently a new discipline, *Synthetic Biology* has emerged which is inspired by systems biology, but pursues a very different philosophy [78]. Where traditional systems biology focuses on understanding of complex living systems through data collection analysis and modelling, the synthetic biology approach aims to better understand biological systems by utilising the building blocks of biological systems to synthesise new systems which may be useful in some way. This approach to biological synthesis is taken at all scales [34], from the modification of DNA molecules to include new synthetic nucleotides and backbones, to the creation of new proteins by altering existing gene sequences or the combining secondary structural elements to replicate the functionality of existing proteins from

scratch and the construction of membrane architectures [218].

At the systems scale, synthetic biology is interested in the creation of new metabolic pathways and networks of gene regulation using approaches to design which will allow the development of systems by the combination of well characterised, composable modules [81]. This modular approach to reasoning about biological systems will provide better understanding of those systems, as the informational and logic processes can be considered without regard of the underlying chemical implementation [200]. Since cellular computing involves the creation within bacteria of logic gates, which are inherently modular and composable, cellular computing and the logic gate abstraction has been integral in the conceptual development synthetic biology systems.

One of the core aspects of synthetic biology is the engineering approach to biological systems [124, 18]. The principle of this approach is to mitigate the problems associated with the design of synthetic regulatory networks (e.g. the risk of crosstalk with the underlying bacterial systems) by utilising modular building blocks, which can be connected together to form new complex systems [81]. As the components of a given module are able to diffuse freely within the cell cytoplasm, which may result in unintended cross-reactions, a means of standardising and characterising biological parts such that a designer of a synthetic system can ignore the underlying details and construct the system using abstract modules is required. A standard for the creation of biological modules known as biobricks, was proposed by [139] and the MIT registry of standard biological parts (`http://partsregistry.org/`) is an attempt to collect and curate a set of modular synthetic biology components, which can be easily interconnected to create new systems.

The synthetic gene regulation systems which have been integrated into bacteria in synthetic biology research are relatively simple, being composed of a few basic components, and adding features such as oscillation and bistability in chemical outputs produced by the bacteria. Gardner et al. were able to demonstrate the construction of a simple switch in E. *coli* based on the Lac repressor [100], the design of which was based on a simple latch, and used negative feedback to generate a bistability. Elowitz and Leibler were able to modify the genome of E. *coli* to include a synthetic oscillator modelled on the repressilator [79]. Wang and Chen detailed the synchronisation of genetic oscillators by the diffusion of signalling molecules amongst populations of cells [261] and in [250] Stricker et al. design and implement a synthetic oscillator which has a tunable period of oscillation. Guantes and Poyatos investigate the dynamical properties of biological oscillation, with the aim of describing a minimal oscillatory architecture [113]. Despite the successful integration of synthetic oscillators into bacterial genomes, there is a notable difference between the reliability and regularity of the synthetic oscillators when compared to natural gene regulation oscillators, such as regulation of gene expression in cyanobacteria by Kai proteins [189]. Later work in this area has shown communication between artificially engineered networks of gene regulation in different bacteria within the same colony, allowing synchronisation of gene expression by an oscillating auto-inducer signal [29], and the implementation of a simple band filter within colonies of bacteria [28].

One of the most ambitious goals of synthetic biology is to better understand what com-

ponents and functionalities are required to support life by creating a minimal organism entirely via methods of artificial synthesis. There are two complementary approaches to creating a minimal cell [124, 78, 251], one approach involves the creation of a synthetic minimal cell from scratch, often termed the "bottom-up" approach, which is explored in more detail in the next section, and the other approach involves the reduction of the genome of an existing organism to a minimal core set of genes required for life, which could then be sequenced and synthesised in the laboratory allowing complete control over the genome contents. The synthetic genome could then be injected into an existing organism containing all the necessary machinery to express the transplanted genome. This "top-down" approach has focused on E. *coli* [141], with deletion strains which had 743 genes removed presented in [215], and the mycoplasma genitalium bacterium, perhaps one of the simplest bacteria. Gibson et al. [102] showed that not only could the entire mycoplasma genome be sequenced, but it could also be reassembled synthetically, and implanted in a living yeast cell.

The endeavour to create a minimal living "chassis" which can be extended with synthetic modular functionality will aid the development of cellular computing, as the reduction in the complexity of microorganism genomes will result in a better understanding of the underlying cellular processes which will provide the platform on which cellular computing will rest. By understanding the underlying systems, the designer of a cellular computing system will be able to anticipate and understand any unexpected interactions between the newly introduced synthetic systems and the chassis. The cell computing paradigm is now considered in more detail in the context of synthetic biology.

2.3.1 Cell Computing

Simple single cellular organisms such as bacteria were first proposed as a possible platform for the creation of biological computing elements by Simpson et al. [240] who considered the possibility of integrating logic gate systems into cells, which they named the "silicon mimetic approach" although the concept was pursued concurrently by Tom Knight, Ron Weiss et al. [265], and numerous other references have drawn on the logic gate analogy to aid understanding of regulatory networks (collections of genes which interact with one another or with other substances in the cell environment in such a way that the rates of transcription of genes in the network are altered). The goal in cell computing is to use bacteria as platforms for the expression of computational behaviour. This may involve the embedding of other forms of biomolecular computing into the bacteria (i.e. by modifying bacteria to perform DNA computation) or by manipulating the bacterial genome to include new genes that act as gene regulatory networks as proposed by Weiss [264]. In this work, the authors attempted to integrate into the bacterial genomes networks of genes which regulate protein expression in a manner which reproduces the functionality of Boolean logic gates. Figure 2.5 illustrates the functionality of a gene regulation based NOT gate in which messenger RNA encoding a transcription factor (a protein or other substance which can bind to a gene sequence and alter its

rate of transcription) acts as the input to the DNA sequence representing a logic gate, the output of which is the mRNA expressed from the logic gate gene sequence. Although these sequences may be artificially integrated into the bacterial genome, work in systems biology has shown that bacterial genomes contain a number of different transcriptional regulation motifs which produce complex functionality [14], and more recently Silva-Rocha et al. [239] performed data mining on prokaryotic promoters and found a large number of logic gate type interactions within the natural bacterial genome.

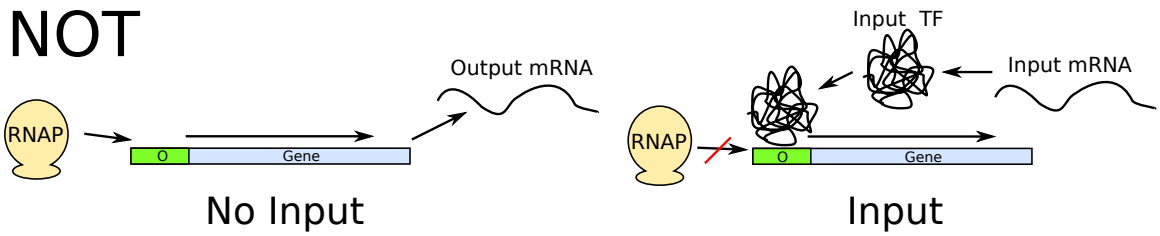


Figure 2.5: A simple NOT gate built from a transcriptional regulation system. The left scenario shows the operation of the gate when no input is present. No input mRNA is present (the input signal is LOW for the gate), and so the RNA Polymerase binds to the operator (O, shown in green) of the gene (shown in blue) and transcribes the output mRNA from the gene sequence, the production of mRNA is equivalent to the logic gate producing a HIGH output. The right scenario shows the operation of the transcriptional NOT gate when input mRNA is present. The input mRNA is transcribed into a transcription factor protein, which binds to the gene operator and prevents the RNA polymerase from binding. Since no output mRNA is produced, the output of the gate is LOW, when the input is HIGH.

Although performance of cellular computing systems is unlikely to rival that of conventional computers, bacteria and other living cells have a number of features which are desirable when attempting to harness biological systems for computation, such as a pre-existing metabolism and the ability to quickly reproduce. As well as this, the information density within a bacterium is many orders of magnitude greater than that which is possible to achieve with silicon based fabrication techniques [240]. A cellular computer could be made to be self-generative, which would mean that vast numbers of computational elements could be grown cheaply and quickly, due to the exponential growth rates of bacteria. Controllable population growth could allow the generation of complicated patterns which could be regenerated if damage to the cellular computer occurred, and reconfigured in response to different stimuli. For example, programmed bacteria could be manufactured as a "smart paint" which could be painted on surfaces and react to certain environmental stimuli. One such example might be painting bacteria onto the surface of a bridge, to detect fatigue or fractures within the material. The bacteria would individually perform a simple set of functions, reproduction in order to ensure that the bacterial computing colony is maintained, and to produce fluorescent protein in the areas where the bacteria detect a chemical stimulus indicating a fracture (e.g. the presence of rust or similar). Genetically engineered bacteria capable of detecting explosives, which function by producing green fluorescent protein (GFP) reporter proteins (which result in the bacteria

glowing green) in the presence of explosive compounds have already been designed and synthesized successfully [244].

The concept of cell computing offers a possible physical implementation of another somewhat more abstract biologically inspired computing paradigms, cellular computing, which was proposed by Sipper et al. [242], and considers the possibility of new computing devices not based on Von Neumann's architecture. Instead computation is performed by the concurrent execution of simple, locally communication devices that exhibit "vast parallelism" (which the author defines as parallel computation involving exponential numbers of processors). The author cites cellular automata, neural networks and DNA computation as being good examples of cellular computation. Cell computing also falls within the definition of *amorphous* computing, [3] in which computation is performed by a large number of simple computational elements, each of which executes a simple program and interacts with other elements executing the same or different programs. The authors consider the difficulties in programming a amorphous computation which must be overcome in order to enable amorphous computing to become a reliable and useful paradigm. In essence, the challenge is to derive emergent computations from systems of simple computing elements which have may exhibit the following problems:

- Some of the devices may be damaged, faulty or inactive.
- The physical placement of devices will likely be irregular.
- The devices may be mobile.
- The devices should produce divergent behaviour despite executing the same program.
- The devices will have very modest computational power and memory.
- Communication between devices will be unreliable and asynchronous.
- No prior knowledge can be assumed about the position or availability of devices.
- Communication between devices can only occur between elements within a local communication radius.

The successful creation of amorphous computing devices will therefore touch on areas of research such as parallel programming, self-assembly, self-organising systems and emergence and communication theory, regardless of the choice of medium for implementation. In an effort to explore the possibilities of the amorphous paradigm computationally, the authors propose a simple programming language for amorphous computing elements, the "growing point language", which is designed to allow straight forward specification of processing, communication and environmental sensing.

Much of the current work towards implementing cell computing in bacteria in the field of synthetic biology has been focussed on the insertion of novel genes and gene networks into Escherichia coli (E. *coli*) bacteria. As E. *coli* is a relatively harmless bacterium that is easily cultured, and has

a genome that is easily manipulated, it has long been a model organism in microbiology, and so forms an excellent platform for the introduction of novel genes and systems. Another commonly studied bacterium is Vibrio Fischeri, which is a species of bacteria known to live in symbiosis with various marine animals, and is of interest as it exhibits quorum sensing and bio-luminescence. In [262], Weiss et al. characterised the behaviour of logic gates implemented in vivo, by considering the requirements for the input and output levels of regulatory network based logic gates in an attempt to ensure that the gates could be duplicated and connected together in sequence. A good overview of the genetic parts which are found in the MIT parts repository and may be useful for the construction of programmable cell computers is given in [259], in which the authors consider regulation by both polypeptides and interference RNA (sequences of RNA which can bind to gene promoters and block transcription), as well as different switch, logic gate and cell-cell communication parts. MIT biobricks were used to design and implement a DNA computing based solution to the "burnt pancake" problem within bacteria in [122], a well known mathematical problem involving sorting by reversals. The hybridisation of DNA computing within a cell computing chassis may prove to be a very successful approach for encoding the solutions of more complex optimisation problems in synthetic biology, as in some respects it combines the benefits of both approaches.

A design for a programmable cell was proposed in [140], in which the synthetic components composing the computation in the cell (the signal sensing, gene regulatory network and output interface) were modular. The authors demonstrated this approach by producing four different strains of E. *coli*, each of which used a different combination of cell signalling pathways (SOS pathway and AHL inducable plasmid), regulatory networks (different plasmids encoding a toggle switch) and output pathways (production of GFP, or a biofilm inducer).

2.4 Bottom Up Synthetic Biology and Vesicle Computing

The alternative "bottom-up" approach to creating a minimal life form is to attempt to synthesise one completely from scratch, using whatever chemical or biological parts and processes are required to construct the synthetic organism on a component by component basis [78, 76]. The creation in the laboratory of a simple entity which is autopoietic, capable of reproduction and reactive to the local environment (often termed a "protocell") which can then be extended with any additional functionality that is required would allow designers of synthetic systems to sidestep many of the issues which arise when manipulating networks of gene regulation in existing organisms, and allow synthetic entities to be constructed from new, non-biological materials. Much of the work in this area takes inspiration from research into the origins of life on earth, as it is thought that a practical approach to the creation of an artificial cell might follow a similar route to the reconstruction of the first cellular life form.

Although there are many possible chemical systems which might be utilised in the creation of an artificial cell, when considering the problem in a more abstract sense, there is a general

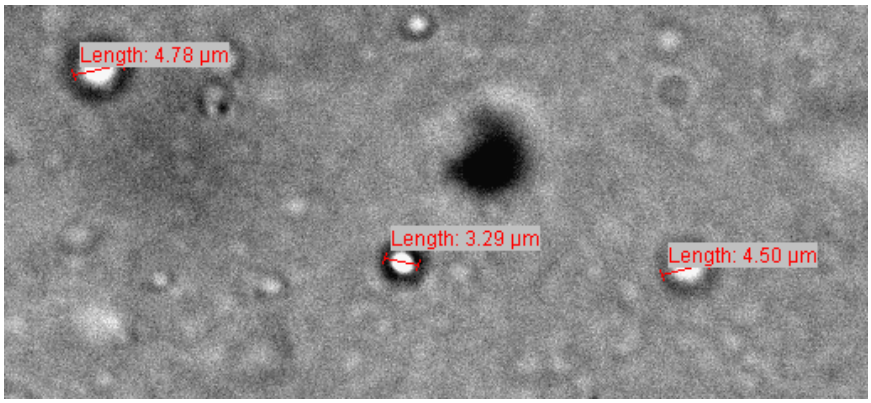


Figure 2.6: Optical microscope images of giant unilamellar vesicles, a measurement indicating approximate diameter is overlayed in red.

agreement over the three components which a minimal life form would require. Firstly, a container is necessary to enable the protocell to control its internal environment, by maintaining concentration gradients between the internal volume and the cell and the external environment. Such a container is also necessary to prevent the uptake of chemicals which would be harmful to the protocell, whilst selectively allowing metabolites to pass into the cell. The encapsulation of the protocell within a small volume also ensures that chemical concentrations can be maintained at the necessary levels to sustain the cell processes. Secondly, a metabolism is required for the cell to be autopoietic, that is to continuously generate the components required to maintain the container, metabolism and other processes, in order to resist the degenerative increase in entropy. Finally, if a protocell is to replicate, then it will require some means of storing the information regarding the manufacture of its components which would permit the automated duplication of the components described. In concert, these properties would likely produce many, if not all of the properties which are associated with life on earth.

Deamer and Pohorille [66, 214] consider one possible approach to the creation of an artificial minimal life-form involving vesicle containers composed of lipid amphiphiles (liposomes) into which DNA and the necessary proteins and chemicals for transcription and translation are placed. Vesicles are spherical structures, made from a membrane composed of amphiphiles, which completely encapsulates an inner volume of liquid and were thought to be integral in the development of early cellular life on earth [67, 68, 252]. Amphiphiles are chemical species which contain both a lipophilic (fat-liking), and a hydrophilic (water-liking) section. The lipophilic section is typically composed of one or more hydrocarbon chains, which do not bond with the surrounding solvent particles, which leads to a disruption of the fluid structure of the surrounding solvent. The hydrophilic group of the amphiphile on the other hand, is typically cationic, anionic, or polar in some way, and will form hydrogen bonds with the surrounding water molecules allowing it to solvate (Figure 2.6 shows an optical microscope image of giant unilamellar vesicles).

As vesicles provide a semi-permeable barrier between the vesicle inner core and the external environment, they can be used as the containers for artificial protocells, which could encapsulate a variety of different chemical and biological systems, including networks of gene regulation, or reactions which can perform computation. The benefit of the bottom-up approach using vesicles is that designers of systems can draw on the huge variety of proteins from existing organisms to provide selected functionality in the model cell. For example [198] showed that it is possible to express heptamer proteins which form pores in the membrane, overcoming one of the issues with vesicles, in that they tend to have membranes which are rather impermeable to charged molecules such as nucleotides.

The expression of current biological proteins within vesicles was first shown by Oberholzer et al. [201], and later a fully encapsulated gene expression system was reported by Nomura et al. [199], in which the authors illustrated the protective nature of the liposome container, by showing that GFP proteins synthesised within the vesicle were not destroyed by proteinase K enzymes placed in the surrounding environment. Nallani et al. [190] created self assembling "synthosomes" , liposomes containing large channel proteins in the membrane. Work by Choi et al. [57, 56] illustrated the possibility of embedding both bacteriorhodopsin and adenosine triphosphate (ATP) synthase proteins within a vesicle membrane, such that the bacteriorhodopsin produced an electrochemical gradient which was shown to drive the ATP synthase protein to convert adenosine diphosphate (ADP) to ATP. Transcription/translation machinery from E. *coli* bacteria was encapsulated within a liposome container and protein expression was observed in [201].

More recent research into the creation of protocells has illustrated the possibility of embedding chemistries relevant to origins of life research, such as the formose reaction [99] within vesicle membranes. The formose reaction is a complex chemical process, which produces various different sugars including ribose, the sugar which forms part of the backbone for DNA and RNA molecules. Chen et al. [52] were able to show the encapsulation within a liposome of an RNA replicase (an RNA molecule capable of catalysing its own reproduction), which adds credence to the possibility that Darwinian evolution may have begun on earth from simple self-replicating strands of RNA (the RNA world hypothesis) and non-enzymatic copying of DNA templates encapsulated within vesicles was shown by Mansy et al. [173]. Encapsulation of an RNA polymerase from the T7 phage within a vesicle was shown by Monnard et al. [186].

As yet the complete design and creation of a protocell exhibiting all three of the cellular life properties described above has remained an elusive goal, despite the claim that the necessary components are already producible in the laboratory [214, 91]. Perhaps one of the most successful attempts was the synthetic bio-reactor created by [198], which was constructed from an artificially assembled vesicle, containing a cell-free extract and a two-stage plasmid based genome encoding a GFP tagged α-hemolysin pore protein. The authors were able to show the expression of the pore protein, which embedded in the vesicle membrane enabling the diffusion across the phospholipid membrane of protocell "food" in the form of nucleotides and amino acids.

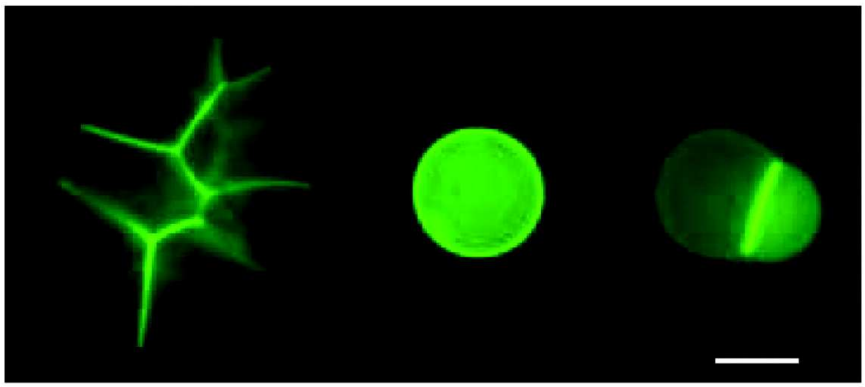


Figure 2.7: Images of vesicle bioreactors (reproduced with permission from PNAS, [198], copyright 2004). The vesicles contain a cell-free extract containing all of the proteins required for transcription and translation. Plasmid DNA encapsulated within the vesicle membrane, and coding for the GFP tagged α-hemolysin pore protein are expressed within the vesicle and porate the membrane. This poration enables nucleotides and amino acids to cross the vesicle membrane, allowing the vesicle to continue translating proteins for over two days.

An even more ambitious approach to the creation of chemical and biological automata from scratch is the attempt to create a *chell* [1], a living organism created entirely from chemicals and processes not found in biology. When attempting to create an artificial life form, it is an unfortunate corollary that to determine the success or failure of such an endeavour, it necessary to subscribe to a fixed definition of what makes something alive (or correspondingly not alive). Clearly, the complete specification of the criteria for a firm definition of life is a problem which has been a topic of biological, philosophical and religious debate for centuries. In order to side-step the necessity of this definition, Cronin et al. take inspiration from Alan's Turing's imitation game, which states that to show an entity is capable of thought, it is only necessary for the entity to imitate intelligence such that it is indistinguishable from an intelligent life form. This argument was applied to the creation of artificial life in Cronin et al. [63], where the authors proposed a test to determine whether a chell was alive, which would be passed if an "interrogator cell", such as a bacterium, was shown to interact with the chell via chemical signalling as if it were another bacterium.

Several large scale initiatives have been created to study the creation of a bottom-up artificial cell in the context of information theory and unconventional computation. The Programmable Artificial Cell Evolution (PACE) project is one such endeavour, which was active from 2004 to 2008. The project was coordinated by Prof. John S. McCaskill and involving collaborators from 13 different institutions located in the EU and the USA. The aim of the PACE project was to investigate the theoretical, technological and experimental challenges relating to a foundation for understanding the information processing capabilities of artificial cells. More recently, the CHELLNet project was an EPSRC funded project investigating the challenges involved in the synthesis of abiotic life-like

[1] a term first used in [63], meaning "artificial chemical cell"

behaviour in complex chemical systems. The work presented in this thesis was created within the context of this project, with the aim of enhancing experimental approaches to the creation of a chell.

The work presented in this thesis is at the intersection between the cellular computing paradigm, the bottom-up approach to creating a minimal life form in the context of synthetic biology, and simulation and modelling techniques. To differentiate this bottom-up approach to the cellular computing paradigm from the bacterial implementations, the concept of using simple protocells encapsulated within vesicles as a platform for the development of a computational protocell is termed *Vesicle Computing*.

2.4.1 Vesicle Computing

The vesicle computing approach proposes that rather than attempting to construct a cellular computing device by modifying existing organisms (as in the top-down approach), many of the problems encountered when modifying bacteria could be avoided by attempting to construct the computing device following the bottom-up philosophy. As well as permitting an engineering approach to the problem, where the complexity of the system is increased as and when new functionality is required and the underlying systems can be fully understood, the bottom-up approach allows the development and application of modelling and simulation techniques for the in silico exploration of the design space.

The creation of a vesicle computer would involve the encapsulation within a vesicle based container of a simple metabolism, which will support a chemical or biochemical information processing system, capable of sensing chemical inputs in the surrounding environment performing simple computation and producing output through chemical signalling. The components for this information processing system could be gene regulatory networks extracted from existing microorganisms, or could be constructed from different chemistries which are not found in biology, or from a combination of both.

Constructing an artificial computational cell from the bottom up has advantages in terms of the materials that are used, as the construction of the system would not be limited purely to biological chemistries, and the cell can be designed to function in environments which may be inhospitable to current microbial species as the designer is not limited to chemistries compatible with life. The creation of programmable protocells is of interest in a number of fields, most especially in pharmacy, where vesicles are proposed as containers for "smart drugs" [40, 74, 75] which may allow controlled release of drugs through the vesicle membrane in-vivo, or store the drug within the vesicle membrane until a relevant disease indicator is detected, which causes the vesicle to rupture, performing a targeted release of the drug in the area where it will have the most effect.

In [38] the authors propose liposomes as containers for the mediation of DNA based biomolecular computing, and recently, Adamatzky et al. proposed the encapsulation of the Belousov-Zhabotinsky (BZ) reaction within hexagonal arrays of vesicles[8, 6]. but to the best of my knowledge,

this thesis presents the first study of a vesicle computing system which approaches the design of the system completely from the bottom-up, harnessing the current research in synthetic biology to propose protocellular computational devices which obey the cellular and amorphous paradigms.

2.5 Simulation and Modelling of Vesicles in Synthetic Biology

One consequence of a successful engineering approach is the decoupling of the design process from the fabrication. In traditional engineering, this is often achieved with the use of computer aided design software, with the proposed design going through an iterative process of modification and testing via simulation *in silico*. Simulation and modelling tools will therefore have an important part to play in the synthesis of biological systems, as once the biobrick components become reliable and predictable enough, it will be possible to simulate the behaviour of a proposed synthetic biology design. It is in this arena, as well as traditional bioinformatics, that computer science can play a significant role in synthetic biology, not only by developing increasing accurate and predictive simulation and modelling techniques, but also by application of more qualitative tools, which allow new designs to be explored in silico, meaning that potential errors in the design can be discovered and addressed before committing to the process of an in-vitro implementation.

2.5.1 The Computational Approach to Biological Modelling

The traditional approach to the modelling of biological systems involves the creation of a set of coupled ordinary differential equations (ODEs), where each equation specifies the change in the concentration of a chemical species present within the system being modelled [65]. For simple models, it may be possible to understand the system dynamics analytically using standard calculus techniques. For more complex models which do not yield to an analytical solution, the behaviour of the system can be observed by performing a numerical analysis of the equations using discrete timestep approximations with integration schemes such as Runge-Kutta. Coupled ODEs are a successful modelling technique in biology and many other disciplines as they are usually computationally inexpensive, accurate when the assumptions made when modelling the system are appropriate, and can be performed using any appropriate ODE solver implementation. Despite these useful qualities, there is a growing consensus that coupled ODEs may not be the best modelling technique when attempting to model the genetic processes occurring with the cell cytoplasm, or when attempting to design new processes from scratch as is often the case in systems and synthetic biology. Instead a more stochastic approach might be appropriate [266]. The argument against ODEs can first be described in terms of assumptions made when using the technique, also from a more philosophical standpoint, and these arguments are now examined in detail.

When developing an ODE model, it is always important to consider the basic assumptions that must be made when using the technique. These assumptions are two-fold. The first is a consequence of the fact that ODEs model the dynamics of numbers in the continuous domain. The

ODEs in a model of a biological system describe the rate at which the concentrations of chemical species change over time due to reactions, degradation etc. which is valid when the volume being simulated is large. However, a problem occurs when considering the tiny volume of the cell cytoplasm, which may be of the order of femtolitres, as concentrations of chemical species at those volumes may represent only a small number of molecules, the assumption that a continuous concentration is a suitable approximation of the system is no longer valid, as concentrations at such volumes may represent fractions of a molecule, which fails to accurately capture the discrete nature of the system.

The second assumption is that the dynamics of the system being modelled are deterministic and are governed solely by the initial concentrations of the chemical species within the volume, and the rate constants. When considering cytoplasmic volumes, the number of molecules of a chemical species may be so small that the stochastic effects caused by Brownian motion become more of a factor in determining the rate of chemical collision (and therefore reaction). Therefore, when considering systems containing positive and negative feedback (such as gene interactions) the deterministic nature of the ODE model is no longer sufficient to capture the behaviour of the system accurately. Shnerb et al. showed that the consideration of the discrete properties of individuals in a spatial model of reproducing cells is very important in determining the macroscopic properties of the population [236], and Gonze et al. compare the behaviour of model circadian oscillators using stochastic and deterministic techniques, showing that stochasticity is important to fully capture the dynamics of the system [107].

More philosophical arguments for the rejection of ODEs as a modelling technique can be made with regards to the highly parallel nature of biological systems. In [217] the author proposes the concept of *algorithmic systems biology*, a concept which is similar to *executable cell biology* [88]. Both of these articles consider the role which computer science may play in the development of systems and synthetic biology organisms. The authors argue that as system biology marks the transition from reductionism to a more system-level understanding of biological phenomena, with greater interest in the mechanistic and causal relationships between systems, the approach to modelling those systems should be algorithmic rather than equation based, considering systems at a higher level of abstraction. Models created using this approach could therefore be programmed in a suitable programming language. Such a change in approach would enable the wide variety of analysis techniques in the field of computer science to be applied to the models, and considers biological systems to be inherently computational. Modelling with algorithms not only requires that the modeller understand and express the underlying mechanisms governing the system behaviour, but also sidesteps the fundamental problems with ODE models described above, as algorithms explore a discrete state space.

Simulation and modelling either traditional or algorithmic, is an aspect of synthetic biology in which computer science can have an important role [61]. Although any programming language could be used to create a model of a biological or chemical system as proposed in algorithmic systems biology, the task becomes easier if the programming language contains features which are relevant

to biological and chemical processes. Numerous theoretical computing models have been proposed which are congruous with the concept of cell computing and algorithmic biology, and these computing models may be useful as abstract programming languages for the configuration of the underlying cell hardware. P systems [210, 211] are one such model, where computations are performed using a discrete representation of chemical reactions within an entire cell or populations of cells In P systems, the computation is expressed in terms of an alphabet of symbols, rules and membranes. Membranes are specified as a hierarchical structure, similar to the membrane and organelle structures found within a cell. Each membrane has assigned to it a multi-set of objects, each of which is a member of the symbol set, and a set of rules which perform a reaction style transition on a set of objects. The computation proceeds by applying the rules assigned to each membrane to the objects in the membrane multi-set in a maximally parallel fashion, such that during one iteration, as many rules as possible for a given combination of objects in the multiset are applied, and rules are applied in all membranes concurrently. More formally, a P system can be represented as the following tuple [211].

$$\Pi = (O, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, i_o) \tag{2.2}$$

Where:

O is an alphabet - its elements are called *objects*

μ is a membrane structure consisting of m membranes, with the membranes (and hence the regions) injectively labelled with 1,2,...,m; m is called the *degree* of Π

$w_i, 1 \leq i \leq m$ are strings which represent multisets over O associated with regions $1, 2, \ldots, m$ of μ

$R_i, 1 \leq i \leq m$ are finite sets of *evolution rules* over O; R_i is associated with the region i of μ; an evolution rule is of the form $u \rightarrow v$, where u is a string over O and v is a string over O_{tar}, where $O_{tar} = O * TAR$ for $TAR = here, out \cup in_j | 1 \leq j \leq m$;

$i_o \in 1, 2, \ldots, m$ is the label of an elementary membrane (the *output membrane*).

The P system computing paradigm includes explicitly some of the most desirable features of biological systems, such as concurrency and parallelism, due to the maximally parallel application of rules, across the membranes in a computation. The topology of a P system is specified as a set of abstract containers, and the application of the rules to the objects drives the computation. The paradigm also includes a number of important features, such as encapsulation, which means that P system computations are modular by design. A large number of different extensions have been made to the original specification of P systems, many of which are inspired in some way by different aspects of cell membrane processes [36, 35, 273, 93, 92, 95, 85, 49]. A similar computational paradigm which considers the processes which can occur in a biological membrane for the purposes of computation called "brane calculi", was proposed by Cardelli et al. [48]. The proposed paradigm

focuses entirely on membrane processes, ignoring the contents of the encapsulated medium, and includes simple actions like encapsulation and dissolution as simple actions in computation.

Despite the desirable properties of these algorithmic approaches, it is still a highly abstracted representation of cellular processes based upon information processing primitives that bear very little resemblance to the chemical processes that are its inspiration. If the P systems model is to be useful as a theoretical tool for designing protocells, the limitations and realities of membrane encapsulated processes occurring at the molecular scale should also be taken into consideration. For example, P systems do not pay any regard to the spatial location, either of membranes (other than by hierarchy of encapsulation) or of objects, as in reality two molecules must be in very close proximity for a reaction to occur. These details are unnecessary for the computational formalism, and this is most likely the reason why they have been removed, but these details are important when creating a formalism that adheres closely to the behaviour of real membrane bound chemical reactions.

Although the above concerns have been discussed in the context of bacterial cell volumes and synthetic biology, simulation of vesicle computing devices should consider the discrete and stochastic nature of the environment, as the volumes of encapsulated fluid in vesicles are typically similar to (or smaller than) those of bacterial cytoplasms. Since vesicle computing is a newly proposed approach, there is no literature regarding the modelling of vesicle computing specifically. However simulation and modelling techniques have been employed to great effect in a related area of study, that of designing and hypothesising protocells. Work in this area is now reviewed.

2.5.2 Modelling Protocell Systems

One key aspect where simulation and modelling can be useful in the design of protocells or vesicle computers, is in the construction and simulation of hypothetical designs of protocellular systems. In simulating a hypothetical system, far more insight is gained into the dynamic functioning of the system than can be taken from a static description. The development of a computational model also ensures that the design is complete and forces the designer to clarify elements of the design which might not be fully explained before modelling.

Gańti's chemoton [97], is an early example of a proposed protocell system which illustrates the utility of simulation and modelling approaches. The chemoton is a simple autopoietic protocellular system which initially contained only a crude metabolism, and the ability to regenerate the components which make up its container. Although the chemoton is perhaps too abstract to be realised in the lab according to its original specification, ODE and stochastic simulation techniques have been used to study the system as a model example of the design of a simple protocellular system. Despite the fact that the chemoton may never be implemented, modelling indicates that such a hypothetical system could be stable, and so the chemoton model acts as a schematic for the implementation of such a system in vitro.

More recently, in [181] Mavelli et al. considered models of chemoton like protocells using the stochastic simulation algorithm, and extended the model to include reproduction of the protocells. This work was further extended by Ruiz-Mirazo et al. [226], in which the authors included the synthesis of the components which make up the protocellular container as part of the metabolism, and in doing so were able to express the reproduction of the protocells in terms of the growth of the membrane and internal volumes, and the resulting changes in membrane surface tension. The modelling of a number of different protocell scenarios was reviewed in [246] and more detailed stochastic models of protocell systems, which include the dynamics of a two dimensional membrane, were studied in [168, 169]. In this model the problem of "protocell starvation" in which the chemicals and components contained within protocells that are constantly dividing become every more diluted, was tackled by including within the model protocell a reaction diffusion system which resulted in a symmetry breaking in the concentrations of encapsulants in the protocell volume, such that when the protocell divides, the daughter cells receive roughly equal concentrations of encapsulant.

If the aim of these protocell designs is to attempt to guide and inform the laboratory creation of such systems, then the simulation and modelling techniques employed should be as detailed as possible within computational constraints. Unfortunately, these constraints currently preclude the use of full scale atomistic techniques such as molecular dynamics (MD) for simulation of entire protocells, although this technique may still play a role in parameter estimation. For vesicle computing and modelling in bottom-up synthetic biology in general, the development of new techniques for simulation at the appropriate scale will be very important, so that a modeller can consider a protocell design at an abstract, systems level as well as being able to consider and test the proposed underlying chemical implementation. In the next chapter, a framework is proposed which connects algorithm modelling techniques such as P systems, with more detailed stochastic modelling techniques, enabling the designer of a vesicle computer to benefit from modelling at several different scales.

2.6 Summary

In this chapter, the development of unconventional computing paradigms has been considered in comparison with the conventional computing paradigm. The research relating to vesicle computing has been discussed in the context of bottom-up research in synthetic biology and the existing approaches to simulation and modelling in this area have been reviewed. The proposed vesicle computing paradigm is at the intersection of three areas of research, unconventional computing, bottom-up synthetic biology and simulation and modelling.

The review of the literature regarding the creation of simple protocells indicates that detailed simulation and modelling techniques may be just as useful in bottom-up synthetic biology as they will be in the attempt to create synthetic biology systems by modifying existing organism genotypes. The focus of this project is to create tools to aid the design and creation of protocells

at the meso scale, by creating a bridge between theoretical descriptions of membrane processes such as P systems or brane calculus, and the real properties of vesicle membranes composed of phospholipid bilayers. To connect P Systems to the properties of in vitro membranes, it is necessary to consider real membranes and to examine how the abstract representations given in P Systems might translate. Since membrane and chemical processes occur on a large, disconnected range of length and timescales, the problem of defining an appropriate simulation methodology for such systems is non-trivial. In the next chapter, a framework for the simulation of vesicle computing experiments at several different length and timescales is proposed and described in detail.

Chapter 3

A Multi-Scale Simulation Framework for Vesicle Computing

In this chapter a multi-scale framework for the simulation and modelling of vesicle computing is described and consideration is given to the time and length scales which are relevant to vesicle simulation. Each of the techniques which are included in the framework is described in detail, and the motivation for using the different simulation techniques which make up the framework is discussed, along with theoretical arguments for their effectiveness at simulating vesicle computing systems.

3.1 Introduction

One of the major challenges in simulation and modelling of complex living systems is determining the correct level of abstraction. The aim of the modeller should be to include as many of the relevant aspects of a system in the model and to remove everything else which is not pertinent to the process being modelled. For example, when considering chemical interaction systems between bacteria such as quorum sensing, it is unnecessary to include in a model of those systems all the details of the internal metabolism of each bacteria, as it is only those anabolised molecules which are directly involved in the process of quorum sensing which are of interest. Practical arguments for the minimisation of complexity in a model can also be stated. Firstly, the goals of modelling include the extraction from the model of insight, information and predictions about the process in question. By introducing unnecessary elements into the model, it may be harder to understand the nature of the process and spot any patterns in the system dynamics that may emerge. Secondly the addition of extra components in the model will generally result in more computationally expensive simulations, which makes an in silico experimental exploration of the simulated system more difficult.

For these reasons, numerous simulation techniques for chemical and physical systems have been proposed, so that the modeller can select the technique which is appropriate for the length and timescales of the process being modelled. Available techniques range from microscopic methods

where every atomic interaction is simulated, to rate equation models which are concerned with bulk concentrations of reactants over long time scales. It is therefore necessary to consider what timescales and length scales are the most important when simulating cellular and vesicle computations, to ensure that the choice of the modelling techniques used is appropriate.

In the vesicle computing paradigm, there is a clear scale separation between the membrane dynamics of the vesicle which typically occur over timescales of nanoseconds to milliseconds and at length scales of angstroms to nanometres and the dynamics of the encapsulated gene regulatory networks which may occur over minutes, hours or days within the vesicle volume. Since this gap in the time and length scales of the processes is so large, no one single simulation method is appropriate for simulating the overall behaviour of the system (Figure 3.1 shows the relevant length and timescales in a vesicle computing simulation). The vesicle computing processes that are of interest can be roughly divided into those that occur at the mesoscale, and those that occur at the system scale. The mesoscale includes processes such as the self-assembly and membrane dynamics of vesicles and bilayers, and the transport properties of chemical signal molecules and is generally defined in terms of nanometre to micrometre length scales and nanosecond to second timescales. Therefore at this scale detailed information is required regarding the molecules which compose the bilayer and the other systems composing the vesicle computer. The system scale is defined here as including those processes which occur over larger length and time scales than those at the mesoscale, but which are of interest when modelling vesicle computing. This includes the degradation of vesicles, vesicle fusion and fission, transcription and translation in prokaryotic genomes, organism development and reproduction. More specifically, the system scale includes processes occurring over time scales from seconds to weeks, and at length scales from micrometres to metres.

The scale separation map in Figure 3.1 indicates that as the processes in vesicle computing span multiple time and length scales, there is no single technique which can be considered as being the most appropriate tool for simulating all aspects of vesicle computation. Instead, a multi scale simulation approach can be employed, with the short timescale/small length scale processes simulated using a technique which includes a high level of detail and the longer timescale processes simulated using a less detailed technique. The simulation techniques which are appropriate for these scales and most relevant to vesicle computing are now reviewed.

3.2 Survey of Available Simulation Techniques

3.2.1 Microscale Simulation Techniques

At the microscale, modelling techniques are available which provide detail all the way down to the atomic scales. Starting at the smallest length and times scales, ab-initio molecular dynamics can be used to simulate the behaviour of molecules based on the quantum effects acting on and between the individual atoms, which provides the most detailed model of molecular interactions, but

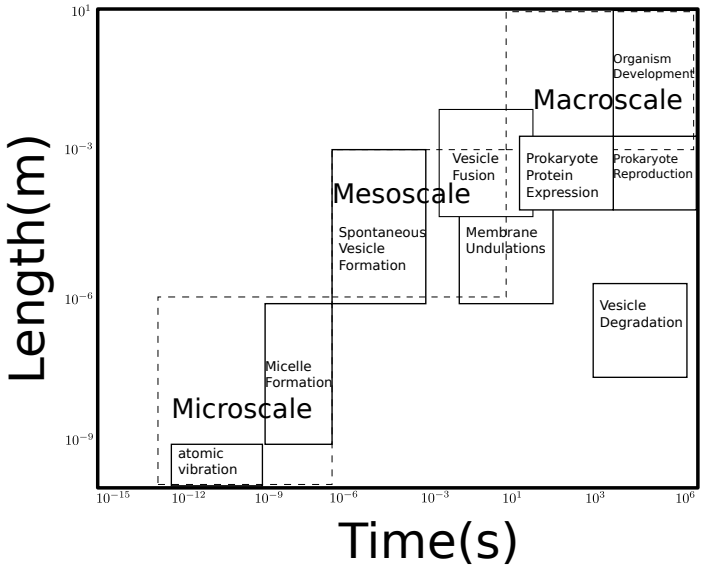
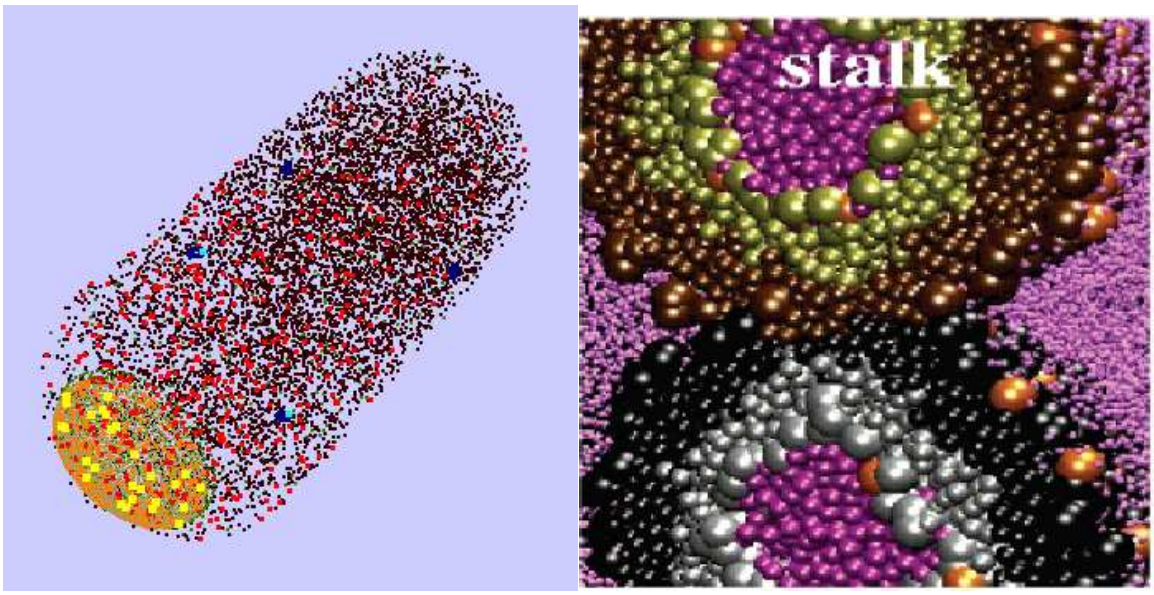


Figure 3.1: Scale separation map for vesicle computing, the figure shows the time and length scales of the processes relevant to vesicle computation, and the length and timescales over which it is appropriate to apply the dissipative particle dynamics and stochastic simulation algorithm techniques.

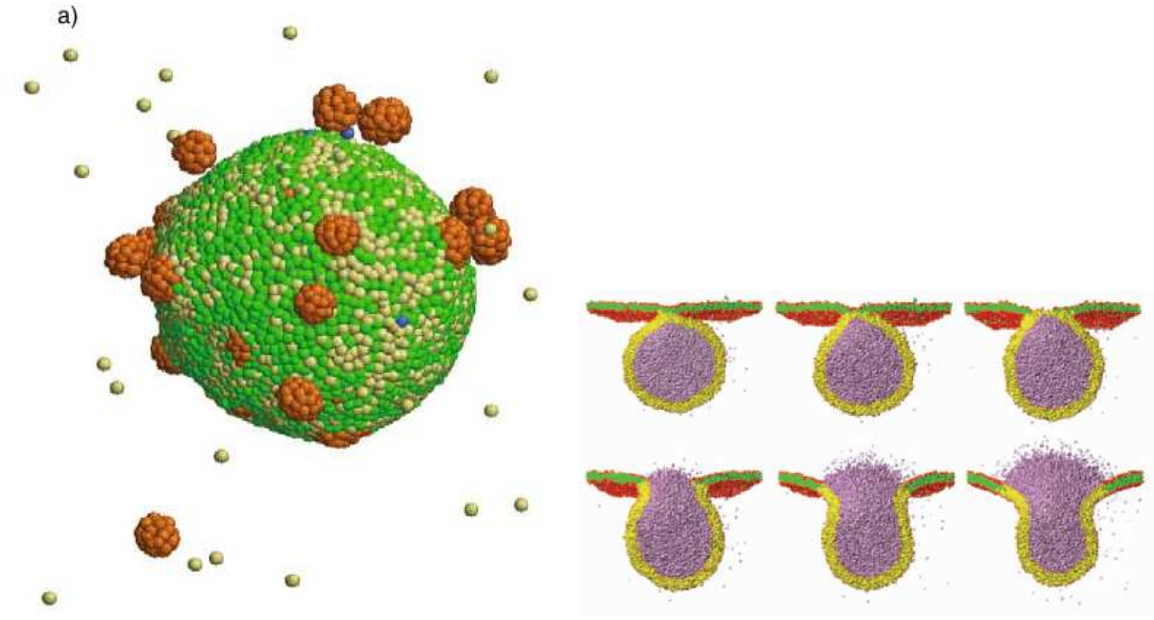
due to the extreme computational demands is limited to simulation of a few hundred atoms [147]. Standard molecular dynamics [219] assumes a system which obeys the laws of classical mechanics, and represents the atomic interactions with the Lennard Jones potential, which is a function that approximates the combined effect of the Pauli exclusion principle and the Van der Waals forces between atoms (Figure 3.2b). Brownian dynamics [83] also employs this potential but removes the requirement for explicit solvent by replacing the solvent interactions with a randomised Brownian noise and dissipative potential which replicates the effect of solvent interaction, enabling simulation of larger length and time scales. The feasible length scales which can be simulated with molecular dynamics can also be extended, by representing the combined effect of a number of atoms of the same chemical element by a single unified atom whilst still maintaining a sufficiently accurate potential, so called "coarse grained" molecular dynamics [225]. The technique has been used for the simulation of membrane dynamics and formation [177].

3.2.2 Mesoscale Simulation Techniques

Mesoscopic particle based membrane simulation techniques typically represent the interactions between a volume of coarse grained particles where each particle may represent one or more atom or molecule of a substance. Polymers are commonly modelled by tethering particles together with simple spring forces, and the interactions between particles are governed by Lennard-Jones style or soft effective potentials which aim to model the net effect of the underlying atomic interactions without resorting to a detailed representation. These techniques can be categorised into those that



(a) (b)



(c) (d)

Figure 3.2: Different microscale and mesoscale simulation techniques: (a) Simulation of E. *coli* metabolic reactions performed with *Smoldyn*, an implementation of the *Smoluchowski dynamics* method [160] (image from `http://www.pdn.cam.ac.uk/groups/comp-cell/Smoldyn.html`). (b) Simulation of vesicle fusion performed with *atomistic molecular dynamics* (reprinted ("adapted" or "in part") with permission from [178] copyright 2003 American Chemical Society). (c) Simulation of vesicle budding due to heterogeneous membrane composition performed using *multipolar reactive Dissipative Particle Dynamics* (reproduced with kind permission from Springer Science+Business Media: [96]). (d) Vesicle fusing with a planar bilayer, performed with *Dissipative Particle Dynamics* (reprinted by permission from Macmillan Publishers Ltd: Nature Materials [232], copyright 2005).

include solvent interaction (explicit solvent models), and those that do not, instead replacing it with potentials representing the effect of the solvent on the atoms/molecules of interest (implicit solvent models).

A number of different explicit solvent methods have been used to simulate lipid bilayers. In [106] for example, Goetz et al. investigate a coarse grained surfactant model for properties such as compressibility and inter-facial tension. In this simulation the parameters for the solvent-amphiphile interaction are determined empirically. A similar study was performed by Imparato et al. [135].

Dissipative particle dynamics [129] is an explicit solvent coarse grained approach, but instead of each particle representing a single molecule, the particles in DPD are fluid elements that represent the average position and centre of mass of a number of molecules of the same type (typically between 3 and 10). The key benefits of DPD over other methods are that the soft potential is short ranged, with gives an order of magnitude performance improvement over coarse grained MD techniques, whilst recreating correctly the hydrodynamics of the simulated system (Figure 3.2d). DPD has been used to simulate a wide variety of soft-matter systems, and has been used to investigate the self-assembly and dynamics of bilayer membranes [111] and phospholipid vesicles [272, 271, 270]. Multipolar reactive dissipative particle dynamics [96] is an extension to the DPD method in which the level of coarse graining is increased further, such that particles represent patches of a bilayer and standard DPD potentials are applied between particles as well as additional potentials which impose a rotational orientation on membrane particles, and impose a curvature on the membrane (Figure 3.2c).

Implicit solvent models have also been used to study membranes at mesoscopic scales. In implicit solvent models the solvent particles are not included in the simulated volume, and are replaced by attractive potentials between amphiphiles which hold the membrane together. The reasoning behind these methods is that the calculation of the equations of motion for large numbers of solvent particles is computationally expensive, and the behaviour of the bulk solvent is uninteresting. Drauffe et al. were able to show the self-assembly of vesicle like structures in a implicit solvent model in which particles represented whole sections of self-assembled bilayer [77]. More detailed simulations of vesicle and membrane formation within a solvent free Brownian dynamics technique have been performed by Noguchi et al. [194, 196, 197, 195].

3.2.3 System Scale Simulation Techniques

Lattice gas automata (LGA) methods are based on the application of cellular automata to the simulation of gas systems. The simplest LGA [120] approach involves the simulation of fluids in a two dimensional lattice in which each lattice site contains up to four gas molecules, and molecules have unit velocity in one of four directions, which represent transitions to the surrounding sites in the lattice. Collisions between molecules occur when two molecules enter the same lattice site with opposite velocities, in which case they are replaced by two particles with velocities at right

angles to those which resulted in the collision. Although the methods can accurately reproduce the Navier stokes equations, and permit straight forward attempts at parallelisation for simulation of large systems, there are several drawbacks to the method. Firstly there is a lack of Galilean invariance and secondly there is a lack of isotropy. In [94] the authors showed that the problem of isotropy could be partly resolved by the modification of the method to use a triangular, rather than square lattice. LGAs were the inspiration for the Lattice Boltzmann method (LBM), and the dynamics are similarly lattice based. However, the key difference between the LBM and LGAs is that the Boolean particle number for each lattice direction is replaced by a continuous "density distribution function" and the collisions between particles replaced by a continuous function known as the "collision operator" [53]. One major drawback when considering lattice based models is the difficulty in simulating polymers in enough detail to investigate the formation of vesicular phases, although some authors have considered the problem of polymer phase separation using the LBM [12] and macroscale studies of vesicle deformation have been reported [155]. Ono et al. had some success in simulation of membrane formation in the presence of chemical reactions within their "artificial lattice chemistry" model, in which membrane assembly was driven by empirically determined forces between membrane particles in the lattice [203, 170].

Smoluchowski dynamics [17] considers molecules as point particles, which diffuse via Brownian motion in a solvent free volume encapsulated by solid boundaries (Figure 3.2a). Collisions in Smoluchowski dynamics may result in reactions, and the method has been used to model a number of cellular processes such as molecular signalling in E. *coli* [161].

For simulation at larger scales, methods generally do not explicitly simulate the trajectory of individual particles, and instead consider higher levels of abstraction, by eliminating the spatial information from the model, and instead introducing equations or rules which correctly recreate the reaction dynamics of systems in solvent. Continuum based simulation methods offer an even greater abstraction for simulation of membranes, in which the details of the amphiphiles which compose the membrane are completely ignored, and instead the membrane is represented by an infinitely thin elastic membrane, with the dynamics governed by the Helfrich Hamiltonian. Using this technique in combination with field theory Ayton et al. performed a simulation of phase separation in a giant unilamellar vesicle membrane [24]. However, continuum simulations typically reduce the level of detail in the simulation to an extent that the technique is no longer viable for the simulation of chemical interactions within vesicle membranes.

The dynamics of the reactions occurring within a cell volume will be stochastic, as Brownian motion of the molecules results in randomised collisions. It has even been suggested that the stochasticity of the cellular environment is key to the survival of life [236]. As ODEs do not involve a random contribution, the integration of an ODE is deterministic, and so the effects of the stochastic environment cannot be represented. Stochastic differential equations (SDEs) offer a solution to this problem, but the properties of the noise term introduced into the equations does not arise from the system being simulated and instead must be specified beforehand, and the technique still suffers

from the issues related to concentration mentioned previously [72].

Discrete stochastic modelling is another category of simulation techniques which overcome the difficulties with ODEs at the expense of increased computational requirements and more complicated algorithms for simulation. These models are exact in the sense that the state of the system is represented by the number of particles of each chemical species, and interactions are represented by the addition (in the case of products) or subtraction (in the case of reactants) from these values. As the occurrence of the reactions is stochastic a simulation using a discrete stochastic technique represents a single path through the possible state space of simulations, and so multiple runs of a single simulation are required to determine the ensemble average behaviour.

One stochastic method which has been very successful in simulating chemical reaction systems is Gillespie's stochastic simulation algorithm (SSA) [104, 103]. The original SSA method is derived from molecular kinetics to produce a theoretically rigorous technique in which each simulation is a Markov chain drawn from distributions which accurately represent the time when the next reaction will occur, and what that reaction will be. Although numerous extensions have been proposed for the SSA algorithm (see [105] for a review) the technique has been used to great effect for the simulation of chemical reaction kinetics in model cell systems in biology [221, 222, 223, 50].

3.2.4 Multi-scale Simulation Techniques

Multi-scale simulation is another possible method which may provide some insights into the dynamics of large vesicles. A technique employed by Ayton and Voth [22, 23] involves simulations of bilayers at three different length scales: the atomic scale with molecular dynamics, the mesoscale with a model similar to DPD and the continuum scale with an elastic model. Parameters for the mesoscale model are then constantly derived from the molecular dynamics simulation during the execution of the simulation, and parameters for the continuum elastic model derived from the dissipative particle dynamics simulation. The authors have applied this technique to the simulation of giant unilammellar vesicle membranes [21] and membrane pore inclusions [25].

Now that the available simulation and modelling techniques for the scales which are relevant to vesicle computing have been described, the framework can be described in more detail and arguments for the choice of techniques employed in the multiscale simulation and modelling framework which is proposed in this thesis can be made.

3.3 The Vesicle Computing Simulation and Modelling framework

The proposed framework supports three distinct stages of modelling. The first of these is the specification of the model using a formal specification language, based on an extended P systems specification language, lattice population P systems (LPP systems). This specification language enables the modeller to describe a model in terms of chemical reactions occurring within membranes, the transport of chemical signal molecules between membranes, and also enables the analysis of

whole colonies of cellular or vesicle computation entities across a lattice of P systems. LPP system specification provides a separation between the specification and the execution of the model, allows modular descriptions of the elements within the model, and supports the specification of parallel systems inherently. By decoupling the specification of the model with the method of its execution, it is possible to use many different techniques to simulate the system. The framework contains three such techniques, DPD, SSA and model checking. Figure 3.3 shows a diagrammatic representation of the vesicle computing simulation pipeline.

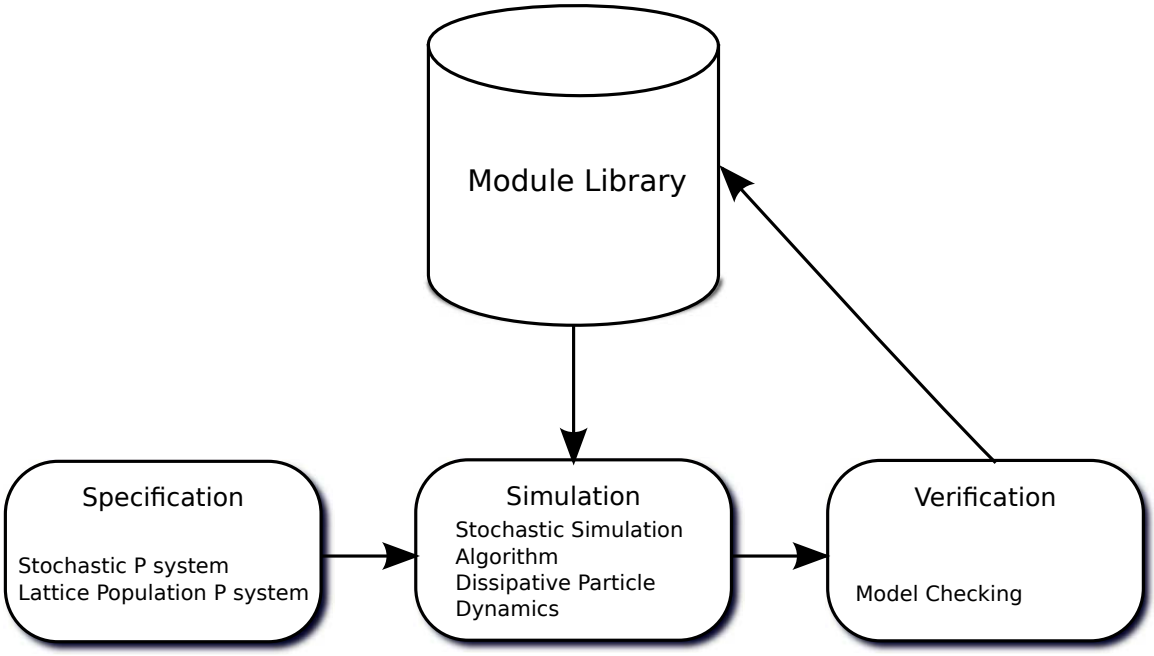


Figure 3.3: The vesicle computing simulation pipeline, the models are specified as stochastic P systems, which can then be simulated using either DPD or LPP systems, depending on the length and time scales which are of interest. The properties of the model can then be investigated using formal model checking techniques.

The second stage of modelling using the framework involves the simulation of the model which has been specified using P systems using either DPD or SSA. The selection of different simulation techniques to perform the simulation of the dynamics of the vesicle computation enables exploration of models over both of the length and timescales described earlier, with mesoscale simulations being performed using DPD, and the system scale dynamics being investigated with SSA. In this way, the modeller can investigate the self-assembly and membrane aspects of a vesicle computation as well as the chemical reaction kinetics. The application of model checking to study the average behaviours of components developed for vesicle computing provides a convenient method for the interrogation of model properties using succinct and formal logic queries.

The final stage of modelling using the framework involves the use of model checking which allows the automated characterisation of properties of the model, enabling the modeller to answer

questions about the model such as "what is the propagation delay of this model logic gate", specified as formal logic queries. If the verification and characterisation of a model indicates that the model will be useful in the future, a module can be specified based on that model which can then be stored in a module library for later use. The P systems based specification language, and the different simulation and model checking techniques used for the modelling of vesicle computing systems are now described in detail.

3.3.1 Model Specification With P systems

P systems are used as a formal specification language falling within the classification of multicompartmental, rule-based, modular and discrete-stochastic modelling frameworks. Although other, equally powerful mathematical languages for modelling concurrently executing systems exist, such as Petri nets and $\pi - calculus$, P systems were selected as they are inspired by the compartmentalisation of reactions and processes in biological cells. Therefore, P systems are the only modelling language which enable specification of computational models in terms of membrane encapsulated regions and rule based transitions which can be used to represent chemical reactions. Furthermore, P systems are the only formalism which explicitly and easily captures the three key properties of single cellular organisms, compartmentalisation, metabolism and transfer of information. In what follows the formal definition of the P systems describing an individual vesicle is presented.

LPP systems are a mathematical formalism which extend P systems and are designed to aid the modelling of cellular systems and bacterial processes, proposed in [223, 243]. In this formalism models are specified as stochastic P systems (SP systems), in which hierarchical structures of membranes each encapsulate a set of reactants and reactions which act on those reactants. The P system model can be duplicated over a lattice structure and communications rules between cells, mimicking the diffusion of small molecules across cell membranes can be defined. LPP systems enable the modeller to focus on the dynamics of reactions with regard to membrane topology over long timescales. LPP systems can be considered to be an extension to the SP systems technique, which adds modelling of encapsulated reactions over a whole population of cells distributed over a spatial grid.

The model is specified in terms of reaction and membrane communication rules as a LPP system, a formal language which allows the expression of a biological model as a topology of membrane encapsulated volumes, with rule sets representing chemical or biological interactions and communication of chemicals within and between those volumes. The specification can be written from scratch or can be used by combining and extending functionalities in the form of "modules" from a module library and this modular approach enables quick and easy specification of complex models from well tested model components. In the next section, stochastic P systems and the lattice population extension are described in detail.

Stochastic P system

An SP system is a formal rule-based specification of a multicompartmental and discrete dynamical system with stochastic semantics given by a tuple

$$\mathcal{SP} = (M, \mu, L, I_{l_1}, \ldots, I_{l_n}, R_{l_1}, \ldots, R_{l_n}) \tag{3.1}$$

where:

- M is a finite set (alphabet) of string-objects specifying the entities involved in the system, molecular species like genes, RNAs, proteins, etc.
- μ is membrane structure composed of $n \geq 1$ membranes defining the regions or compartments of the system. Membranes are arranged in a hierarchical manner, i.e. membranes can be inside other membranes. There exists an outermost membrane termed the *skin*, not contained in any other membrane, which defines the system itself acting as the boundary. The membrane structure can be represented graphically using a Venn diagram, a list of matching square brackets or formally as a rooted tree where each node represents a membrane, the root stands for the skin membrane and the relationship of a membrane being inside another one is described by the node representing the first membrane being the descendant of the node for the second membrane.
- $L = \{l_1, \ldots, l_n\}$ is finite set of labels naming compartments in a one-to-one manner, i.e. nucleus, cytoplasm, etc.
- I_{l_k} for each $1 \leq k \leq n$ is the initial condition of the compartment or region defined by membrane k consisting of a multiset of objects over M describing the initial number of the different entities, molecular species, present in the corresponding compartment at the initial state of the computation or evolution of the system.
- $R_{l_k} = \{r_1^{l_k}, \ldots, r_{m_{l_k}}^{l_k}\}$ for each $1 \leq k \leq n$ is a set of multiset rewriting rules describing the interactions between the entities of the system, molecular interactions like complex formation, gene regulation etc. Each set of rewriting rules R_{l_k} is specifically associated to the compartment identified by the label l_k. These multiset rewriting rules are of the following form:

$$r_i^{l_k} \ : \ o_1 \ [\ o_2 \ ]_l \overset{c_i^{l_k}}{\rightarrow} o_1' \ [\ o_2' \ ]_l \tag{3.2}$$

  where o_1, o_2 and o_1', o_2' are multisets of objects (possibly empty) over M representing the reactants and products in the corresponding molecular interaction. The square brackets and the label l describe the compartment involved in the interaction. An application of a rule of this form changes the content of the membrane with label l by replacing the multiset o_2 with

o'_2 and the content of the membrane outside by replacing the objects o_1 with o'_1. A stochastic constant $c_i^{l_k}$ is specifically associated with each rule in order to determine the probability of applying the rule and the time elapsed between rule applications according to the stochastic semantics of the simulation method being used.

This definition provides the formalism needed for the specification of the reactions, topology and membrane dynamics of an individual model vesicle.

The specification of complicated models using SP systems is aided by the provision of modules [223]. A module is a discrete element of functionality, composed of rules which act on a set of chemical species which are not defined in the module's specification, but instead can be fixed by passing in a set of chemical species identifiers to the module when using it as a building block within a model design. Modules can be considered as meta-rulesets, which specify the interactions between chemical species, but do not specify what those chemical species are. Modules can be used to represent the motifs that appear repeatedly in transcriptional networks [15]. SP system modules are also congruous with the synthetic biology approach to the design of novel transcriptional networks, in which designs will be produced by combining well characterised and standardised functional building blocks, as exemplified in the MIT Biobricks project [46, 230]. The concept of an SP system module is now introduced in more detail.

P system Module

A P system module is identified with a name, Mod, and three finite ordered sets of variables $O = \{O_1, \ldots, O_n\}$, $C = \{C_1, \ldots, C_m\}$ and $Lab = \{L_1, \ldots, L_o\}$ and it consists of a finite set of rewriting rules of the form in (3.2):

$$Mod(O, C, Lab) = \{r_1, \ldots, r_p\} \tag{3.3}$$

The objects, stochastic constants and labels of the rules in module Mod can contain variables from O, C or Lab respectively. These variables can be instantiated with specific molecular species names, numerical values for the stochastic constants and compartment names. The instantiation of a module $Mod(O, C, Lab)$ with specific values $o = \{o_1, \ldots, o_n\}$, $c = \{c_1, \ldots, c_m\}$ and $lab = \{l_1, \ldots, l_p\}$ for O, C and Lab respectively is represented as:

$$Mod(\{o_1, \ldots, o_n\}, \{c_1, \ldots, c_m\}, \{l_1, \ldots, l_p\}) \tag{3.4}$$

the rules are obtained by applying the corresponding substitutions $O_1 = o_1, \ldots, O_n = o_n$, $C_1 = c_1, \ldots, C_m = c_m$ and $L_1 = l_1, \ldots, L_o = l_o$.

The definition of the *P system module* allows the hierarchical description of a complex module, $M(O, C, Lab)$, by obtaining its rules as the set union of simpler modules, $M(O, C, Lab) = M_1(O_1, C_1, Lab_1) \cup \cdots \cup M_n(O_n, C_n, Lab_n)$ with $O = O_1 \cup \cdots \cup O_n$, $C = C_1 \cup \cdots \cup C_n$ and $Lab =$

$Lab_1 \cup \cdots \cup Lab_n$. Finally, the set of rules, R_{l_k}, in *SP systems* can be specified in a modular way as the set union of several instantiated *P system modules*, $R_{l_k} = M_1(o_1, c_1, lab_1) \cup \cdots \cup M_{n_k}(o_{n_k}, c_{n_k}, lab_{n_k})$.

The use of modularity allows the definition of libraries or collections of modules:

$$Lib = \{Mod_1(O_1, C_1, Lab_1), \ldots, Mod_m(O_m, C_m, Lab_m)\} \tag{3.5}$$

This modular approach aids the specification of vesicle computing models in two ways. Firstly modules from different libraries can be instantiated with multiple specific molecular species names, stochastic constants and compartment names so they can be reused in different SP system models of vesicle systems. Secondly, modularity reduces the amount of syntax required when specifying a model, enabling the designer to consider systems at a higher level of abstraction. The ability to specify modules in the specification of vesicle computing systems should allow the modeller to easily specify large and highly complicated models by breaking them up into a combination of modules, making models easier to specify, interpret and modify. The benefits of a modular approach are analogous to those of programming a computer using an object oriented programming language, as it permits design at a higher level of abstraction, and encourages the development of reusable, self-contained elements of functionality.

Module	**Rules**	**Description**
$Const(\{geneX, rnaX\},$ $\{c_1\}, \{b\})$	$[\ geneX\ ]_b \xrightarrow{c_1} [\ geneX + rnaX\ ]_b$	Constitutive gene expression
$Pos(\{protY, geneX, rnaX\},$ $\{c_1, c_2, c_3\}, \{b\})$	$[\ protY + geneX\ ]_b \xrightarrow{c_1} [\ protY.geneX\ ]_b$ $[\ protY.geneX\ ]_b \xrightarrow{c_2} [\ protY + geneX\ ]_b$ $[\ protY.geneX\ ]_b \xrightarrow{c_3} [\ protY.geneX + rnaX\ ]_b$	Positive gene regulation
$Neg(\{protY, geneX, rnaX\},$ $\{c_1, c_2, c_3\}, \{b\})$	$[\ protY + geneX\ ]_b \xrightarrow{c_1} [\ protY.geneX\ ]_b$ $[\ protY.geneX\ ]_b \xrightarrow{c_2} [\ protY + geneX\ ]_b$ $[\ geneX\ ]_b \xrightarrow{c_3} [\ geneX + rnaX\ ]_b$	Negative gene regulation
$PostTransc(\{rnaX, protX\},$ $\{c_1, c_2, c_3\}, \{b\})$	$[\ rnaX\ ]_b \xrightarrow{c_1} [\ \ ]_b$ $[\ rnaX\ ]_b \xrightarrow{c_2} [\ rnaX + protX\ ]_b$ $[\ protX\ ]_b \xrightarrow{c_3} [\ \ ]_b$	Post-transcriptional regulation

Table 3.1: Library of modules for basic transcriptional and post-transcriptional regulation

An example of a module library is presented in Table 3.1. This library contains modules modelling the basic regulatory mechanisms in gene transcription and post-transcriptional regulation. Table 3.2 presents some example SP system models of three recurring patterns in gene regulation networks, namely, constitutive expression, positive auto-regulation and negative auto-regulation of a gene, which are constructed from the modules in Table 3.1.

The SP system models can be distributed over a spatial lattice, an array of regularly

$$\mathcal{SP}_{UnReg} = (M_{UnReg}, \mu_{UnReg}, L_{UnReg}, I_{bact}, R_{bact}) \quad \text{where}$$
$$M_{UnReg} = \{geneTf, rnaTf, Tf\}$$
$$\mu_{UnReg} = [\ ]$$
$$L_{UnReg} = \{bact\}$$
$$I_{bact} = \{gene\}$$
$$R_{bact} = \begin{cases} Const(\{geneTf, rnaTf\}, \{0.025\}, \{bact\}) \cup \\ PostTransc(\{rnaTf, Tf\}, \{0.07, 3, 0.01\}, \{bact\}) \end{cases}$$

$$\mathcal{SP}_{PAR} = (M_{PAR}, \mu_{PAR}, L_{PAR}, I_{bact}, R_{bact}) \quad \text{where}$$
$$M_{PAR} = \{geneTf, Tf.geneTf, rnaTf, Tf\}$$
$$\mu_{PAR} = [\ ]$$
$$L_{PAR} = \{bact\}$$
$$I_{bact} = \{gene\}$$
$$R_{bact} = \begin{cases} Const(\{geneTf, rnaTf\}, \{0.0025\}, \{bact\}) \cup \\ Pos(\{Tf, geneTf, rnaTf\}, \{1, 0.6022, 0.025\}, \{bact\}) \cup \\ PostTransc(\{rnaTf, Tf\}, \{0.07, 3, 0.01\}, \{bact\}) \end{cases}$$

$$\mathcal{SP}_{NAR} = (M_{NAR}, \mu_{NAR}, L_{NAR}, I_{bact}, R_{bact}) \quad \text{where}$$
$$M_{NAR} = \{geneTf, Tf.geneTf, rnaTf, Tf\}$$
$$\mu_{NAR} = [\ ]$$
$$L_{NAR} = \{bact\}$$
$$I_{NAR} = \{gene\}$$
$$R_{bact} = \begin{cases} Neg(\{Tf, geneTf, rnaTf\}, \{1, 0.6022, 2\}, \{bact\}) \cup \\ PostTrasnc(\{rnaTf, Tf\}, \{0.07, 3, 0.01\}, \{bact\}) \end{cases}$$

Table 3.2: Three simple SP system models of unregulated (constitutive) expression, positive auto-regulation and negative auto-regulation of a gene, instantiated in a container labelled *bact*.

distributed points in two dimensions, which describes the topology of the population of cells. This enables the modelling of parallel vesicle computation, in which each cell in the lattice holds an SP system which is a model of a vesicle computer.

Lattice Definition

A two dimensional lattice *Lat* can be specified by the integer bounds $(\alpha_1^{min}, \alpha_1^{max}, \alpha_2^{min}, \alpha_2^{max})$, and each point on the lattice is uniquely identified by two integer coordinates denoted as $x = (\alpha_1^{min} \leq \alpha_1 \leq \alpha_1^{max}, \alpha_2^{min} \leq \alpha_2 \leq \alpha_2^{max})$.

Populations of vesicle computers are distributed on the lattice and each individual vesicle computer model is specified in a modular fashion as an SP system (Equation 3.1), and different copies of the corresponding SP system representing each vesicle computing model are distributed over the points in the lattice according to the spatial distribution of the cells in the population. Populations of vesicle computers are therefore specified as *lattice population P systems*.

Lattice Population P system

A lattice population P system (LPP system) is a formal specification of an ensemble of cells distributed according to a specific geometric disposition given by the following tuple:

$$\mathcal{LPP} = (Lat, (\mathcal{SP}_1, \ldots, \mathcal{SP}_p), Pos, (\mathcal{T}_1, \ldots, \mathcal{T}_p)) \tag{3.6}$$

where

- Lat is a lattice describing the topology of the ensemble of cells.
- $\mathcal{SP}_1, \ldots, \mathcal{SP}_p$ are SP systems as in Equation 3.1 specifying the different cell types in the population.
- $Pos : Lat \rightarrow \{\mathcal{SP}_1, \ldots, \mathcal{SP}_p\}$ is a function distributing different copies of the SP systems $\mathcal{SP}_1, \ldots, \mathcal{SP}_p$ over the points of the lattice.
- $\mathcal{T}_k = \{r_1^k, \ldots, r_{n_k}^k\}$ for each $1 \leq k \leq p$ is a finite set of rewriting rules termed *translocation rules* that are added to the skin membrane of the respective SP system $\mathcal{SP}_k$ in order to allow the interchange of objects between SP systems located in different points in the lattice. These rules are of the following form:

$$r_i^k : [\ obj\ ]_l \overset{\mathbf{v}}{\bowtie} [\ ] \overset{c_i^k}{\rightarrow} [\ ]_l \overset{\mathbf{v}}{\bowtie} [\ obj\ ] \tag{3.7}$$

where obj is a multiset of objects, $\mathbf{v}$ is a vector in $\mathbb{R}^n$ and c_i^k is the stochastic constant used in the algorithm to determine the dynamics of rule applications. The application of a rule of this form in the skin membrane with the label l of the SP system $\mathcal{SP}_k$ located in the point

$\mathbf{p}$, $Pos(\mathbf{p}) = \mathcal{SP}_k$, removes the objects obj from this membrane and places them in the skin membrane of the SP system $\mathcal{SP}_{k'}$ located in the point $\mathbf{p} + \mathbf{v}$, $Pos(\mathbf{p} + \mathbf{v}) = \mathcal{SP}_{k'}$.

An example of an LPP system consisting of a rectangular finite point lattice where copies of the positive auto-regulation SP system (presented in Table 3.2) are distributed is shown in Figure 3.4. In the next section, a detailed description is given of the technique which will be used for the more detailed simulations of vesicle computing, dissipative particle dynamics.

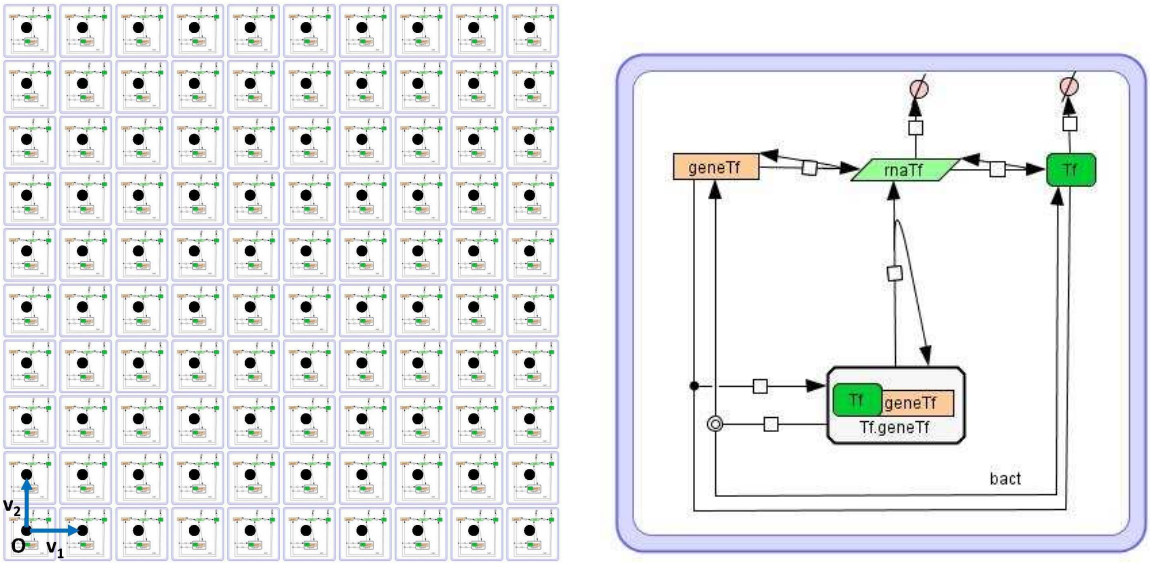


Figure 3.4: A lattice population P system consisting of SP systems distributed over a lattice $Lat = (0, 10, 0, 10)$. The SP system representing gene positive auto-regulation introduced in Table 3.2 is graphically represented on the right and is distributed over all the positions of the lattice. Notice that there is no communication between the cells in this lattice.

Stochastic and lattice population P systems are the formalisms used for the specification of vesicle computing models, which can then be simulated using either the dissipative particle dynamics or the stochastic simulation algorithm. These simulation techniques are now described in detail, starting with the mesoscale dissipative particle dynamics technique, which is appropriate for the simulation of individual vesicle P systems at a high level of detail.

3.3.2 Mesoscale Simulation With Dissipative Particle Dynamics

Dissipative particle dynamics (DPD) is a coarse grained particle dynamics simulation technique first proposed in [129]. In a typical DPD simulation, a large number of particles are distributed within a three dimensional volume, short range soft interaction potentials are defined between particle pairs, and the equations of motion are numerically integrated. The particles in the simulation do not represent individual atoms or molecules and instead represent the average position of a number of molecules of a chemical species. Particles may also have spring and angle forces applied between

them, to produce model polymers. In essence the modelling approach in DPD is to represent a continuous volume of fluid as discrete "chunks" of matter, the DPD particles. The original DPD technique was described by [129] and was intended as an off lattice solution to the Navier Stokes equation. DPD was given a foundation in statistical mechanics by Espanol et al. [84] who derived the Fokker-Plank equation for the method. The mapping between the particles and physical molecules was investigated by Groot and Warren [112], who determined the length and time scales of the simulation in relation to water molecules.

In a particle dynamics simulation, particles have explicit position and velocity and the simulation is numerically integrated in discrete steps which represent the passage of time. The laws of classical mechanics apply to particles in a particle dynamics simulation, and so the change in a particle's velocity is related to the forces acting upon it and the particle's mass. The potentials between particles are typically defined to represent the interactions between the physical objects that the particles represent (e.g. atoms or molecules). In the case of molecular dynamics, a number of different potentials may be employed, but the most common is the Lennard Jones potential, which models the combination of the Pauli exclusion principle which prevents the electron orbitals of atoms from overlapping, and the attractive potential between bonded atoms described by the Van der Waals interaction. If ionic interactions are required, then the Coulomb potential may be calculated as well. Intra-molecular bonding potentials are also added to maintain a molecular structure.

As DPD particles represent the average position of a number of molecules rather than single atoms, the Lennard Jones potential is no longer an appropriate representation of the interactions between particles. Instead, the DPD method employs a simple repulsive force between particles which is inversely linear to the distance between the two particles up to a short-ranged cut-off. If the conservative force was the only one included in the DPD method, then the simulated particles would reach an equilibrium point where the repulsive forces acting on each particle are minimised, and the system would freeze. In order to replace the contribution of the internal energy of the molecules, which is lost due to the coarse graining, the DPD method includes two further forces, the random force and the dissipative force.

Pairs of DPD particles interact via three different potentials, which act along the line of centres of the two particles. The random and dissipative potentials control the temperature of the simulation, by introducing and removing energy from the simulation and the conservative force introduces a parameterised repulsion between particles. For a pair of particles i and j, where the distance between them r_{ij} is less than the force interaction radius r_c, the total force acting on each particle F_{ij} is the summation of the conservative, dissipative and random forces acting on those particles. Each of the three forces conserves linear and angular momentum.

$$F_{ij} = F_{ij}^C + F_{ij}^D + F_{ij}^R \tag{3.8}$$

$$F_{ji} = -F_{ij} \tag{3.9}$$

The conservative force produces a parameterisable repulsion between different particle types.

$$F_{ij}^{C} = a(1 - \frac{|r_{ij}|}{r_c}) \tag{3.10}$$

where the a parameter is the maximum strength of the conservative force for the two particle types, r_{ij} is the vector pointing from particle j to particle i, and r_c is the force interaction radius. Two particles experience a further repulsive force if they are moving towards one another, and a drag force if they are moving apart. This effect is created by the dissipative force.

$$F_{ij}^{D} = -\gamma w^{D}(r_{ij})(\hat{\mathbf{r}}_{ij} \cdot \mathbf{v}_{ij})\hat{\mathbf{r}}_{ij} \tag{3.11}$$

where the γ parameter controls the maximum strength of the dissipative force, $\hat{\mathbf{r}}_{ij}$ is the unit vector pointing from particle j to particle i and v_{ij} is the relative velocity between particle j and i. w^D is the dissipative weighting function, described below. The random force produces a random repulsion or attraction between particle pairs.

$$F_{ij}^{R} = \frac{\sigma w^{R}(r_{ij})\theta_{ij}\hat{\mathbf{r}}_{ij}}{\sqrt{dt}} \tag{3.12}$$

where the σ parameter controls the maximum strength of the random force, θ_{ij} is a random variable with zero mean and unit variance, and dt is the length of the simulation time step. w^R is the random weighting force, described below. Two further forces are added between particles to model polymerisation between particles. The first is a simple Hookean spring force applied between particle pairs which represents covalent bonding.

$$F^{bond} = k(l_0 - |r_{ij}|) \tag{3.13}$$

Where k is the maximum bond force constant, and l_0 is the preferred bond length. Bond angles are imposed by a potential acting between three particles.

$$F^{angle} = sin(\theta_{ijk} - \theta_0) \cdot k_\theta \tag{3.14}$$

where θ_{ijk} is the interior angle between particles i, j and k, θ_0 is the preferred bond angle and k_θ is the maximum force magnitude.

The coarse graining approach taken with the DPD method allows simulation of systems at the mesoscale, which is highly computationally expensive with traditional molecular dynamics methods. The consequence of this coarse graining is the averaging out of many of the inter-atomic interactions, which can introduce difficulties and artifacts into the simulation results which the modeller should be aware of. Firstly, the DPD method cannot be used to simulate liquid-vapour phase boundaries [112]. Also, the soft core potentials used in DPD mean that the Schmidt number

(the ratio between the kinematic viscosity and the mass diffusivity) is typically close to unity, which is unrealistic for real fluids in which this number of typically 4 or 5 magnitudes larger. This is due to the soft potentials, which do not replicate the "caging" effect of the surrounding environment on a fluid molecule and so mass diffusion occurs almost at the same rate as momentum diffusion [112]. Another consideration when using DPD for membrane simulations, is that the elasticity of the membrane is typically 50%-60% larger than the experimentally observed values, although recently improved parameterisations of the lipids have been proposed which counter this problem [98]. Despite these issues DPD has become a widespread and useful tool for investigation of membrane properties.

As the DPD method is composed of an inter-particle force (F^C), a viscous force (F^D) and a random force representing Brownian motion (F^R), the DPD method can be expressed as a Langevin equation, which can then be expressed as a stochastic differential equation. To give DPD a firm grounding in statistical physics, the distribution function $\rho(r_i, p_j, t)$ can be derived which gives the probability of finding the system in a state where the particles have positions r_i and momenta p_j at any particular time t. The time evolution of this distribution is governed by the Fokker-Planck equation and this equation was derived for the DPD method in [112].

When the system is in equilibrium, $\partial_t \rho(r_i, p_j, t)$ should be equal to zero, and in order for this to be the case Espanol [84] showed that the force parameters and the weighting functions for the dissipative and random forces should be related as follows.

$$\sigma = \sqrt{\gamma 2 k_B T} \tag{3.15}$$

$$W_D(r) = [W_R(r)]^2 \tag{3.16}$$

The weighting function can be any function of inter-particle distance, but is typically set as follows.

$$w^R(r) = \begin{cases} (1-r) & when(r < 1) \\ 0 & \text{otherwise} \end{cases} \tag{3.17}$$

Setting the parameters and weighting functions in this way ensures that $\rho(r_i, p_j, t)$ does not move away from equilibrium.

Setting the DPD Parameters

The modelling of vesicle systems in DPD involves the specification of the parameters described above, and a proposed bond structure for the polymers which will form the bilayer. The parameters for the system should be set such that the resulting dynamics resemble those of the system being modelled as closely as possible. The most important parameter in terms of calibration of the simulated system with the modelled one, is the conservative force a parameter matrix, which specifies the repulsive force strength between particle types. The parameter relates directly to the miscibility of simulated fluids.

Groot and Warren proposed a method for setting the conservative force a parameter in DPD for the modelling of homogeneous fluids [112], which involves the calibration of an observable value between the simulated fluid and the fluid being modelled. The authors choose the dimensionless compressibility.

$$\kappa^{-1} = \frac{1}{nk_BT\kappa_T} \tag{3.18}$$

$$= \frac{N_m}{k_BT}\left(\frac{\partial p}{\partial n}\right)_T \tag{3.19}$$

Where κ_T is the isothermal compressibility, and n is the number density of the molecules (or beads in DPD), and N_m is the bead number (i.e. the number of water molecules that a DPD particle represents). To establish a correspondence between the simulated and actual value, the authors determine the pressure in the simulation from the density, using the virial theorem.

$$p = \rho k_BT + \frac{1}{3V}\langle \sum_{j>i} (r_i - r_j) \cdot F_{ij}^C \rangle \tag{3.20}$$

and then determined the equation of state for DPD. For densities which were sufficiently high ($\rho > 2$) this equation is:

$$p = \rho k_BT + \alpha a \rho^2 (\alpha = 0.101 \pm 0.001) \tag{3.21}$$

Where a is the conservative force parameter. This indicates that there is a simple scaling relationship between system pressure and the a parameter value. Using equation 3.19, the relation between the dimensionless compressibility and the alpha parameter can be derived.

$$\kappa^{-1} = 1 + 2\alpha a\rho / k_BT \tag{3.22}$$

By combining equation 3.22 with the known dimensionless compressibility of water ($\kappa_{water}^{-1} = 15.9835$) it can be shown that $a\rho/k_BT$ is roughly 75. It is then only necessary to set the density, as the aim is to minimise the number of simulated particles, therefore enabling simulation of larger systems. The smallest density at which the equation of state still holds is a good choice, and so $\rho = 3$. The alpha parameter for a simulated fluid which reproduces the compressibility of water is then $a = 25k_BT$. The conservative force parameter for other fluids is also easily determined for other fluids, as long as the compressibility is known.

The above parameterisation is sufficient to reproduce compressibility in simple homogeneous fluids. For more complex systems of mixing fluids, Groot and Warren were able to show a relationship between the DPD conservative force parameter, and the well understood Flory-Huggins χ interaction parameter. The Flory-Huggins theory of polymers considers the mixing of polymers which are confined to a lattice [131, 132, 89]. Each site in the lattice is either filled with a solvent molecule or a monomer, and all lattice sites are filled. The theory can be used to estimate the free

energy of mixing for a polymer. The DPD method and the Flory-Huggins model are quite similar, and DPD can be considered to be a continuous version of the Flory-Huggins model [112]. The mixing of components can be thought of in terms of change in entropy, which can be calculated according to Boltzmann's relation

$$\Delta S_m = k \ln \Omega \tag{3.23}$$

where k is Boltzmann's constant and Ω is the number of microstates consistent with a given macrostate. By discretising the continuous space into a lattice, it becomes possible to calculate directly the number of microstates for a given number of molecules in a Flory-Huggins lattice. For a system with a macrostate with n_1 components of type 1, and n_2 components of type 2, with the total number of components (and lattice sites) given by $N = n_1 + n_2$, Ω can be calculated by simple combination.

$$\Omega = \binom{N}{n_1} \tag{3.24}$$

$$= \frac{N!}{n_1! n_2!} \tag{3.25}$$

Since the coefficients in the above equation will be large, the Stirling approximation can be used, leading to the change in entropy for the mixing of the two components.

$$\Delta S_m = -k(n_1 \ln x_1 + n_2 \ln x_2) \tag{3.26}$$

where $x_1 = n_1/(n_1+n_2)$ and $x_2 = n_2/(n_1+n_2)$. For polymers, this entropy is lower as the restriction of the movement of the monomers reduces the possible number of microstates. If we now consider n_1 and n_2 to be two different polymers composed from r_1 and r_2 monomers respectively, then the adjusted entropy calculation is as follows:

$$\Delta S_m = -k(n_1 \ln \phi_1 + n_2 \ln \phi_2) \tag{3.27}$$

where ϕ_1 and the ϕ_2 are the volume fractions of the two polymers, calculated as follows.

$$\phi_1 = \frac{r_1 n_1}{r_1 n_1 + r_2 n_2} \tag{3.28}$$

$$\phi_2 = \frac{r_2 n_2}{r_1 n_1 + r_2 n_2} \tag{3.29}$$

Since the above equation gives the free energy for the mixing over the whole lattice, and the quantity of interest is the free energy per interaction site (which corresponds to the free energy per particle

in DPD), it is necessary to divide by the total number of lattice sites.

$$\frac{\Delta s_m}{N} = -k_B(\frac{n_1}{N}\ln\phi_1 + \frac{n_2}{N}\ln\phi_2) \tag{3.30}$$

$$= -k_B(\frac{\phi_1}{r_1}\ln\phi_1 + \frac{\phi_2}{r_2}\ln\phi_2) \tag{3.31}$$

The Gibbs free energy of mixing for a lattice can be derived from the enthalpy and entropy as follows:

$$\Delta G_m = \Delta H_m - T\Delta S_m \tag{3.32}$$

The enthalpy can be expressed in terms of the Flory Huggins interaction parameter χ.

$$\frac{\Delta h_m}{k_B T} = \chi\phi_1\phi_2 \tag{3.33}$$

Where χ is defined as

$$\chi = \frac{z\Delta\omega_{12}}{k_B T} \tag{3.34}$$

where $\Delta\omega_{12} = \omega_{12} - \frac{1}{2}(\omega_{11} + \omega_{22})$, the change in internal energy for formation of an unlike molecular pair and ω_{ij} is the energy of i to j contacts, and z is the number of cell neighbours. The equation for the free energy in a single lattice site is then

$$\frac{\Delta g_m}{k_B T} = \frac{\phi_1}{r_1}\ln\phi_1 + \frac{\phi_2}{r_2}\ln\phi_2 + \chi\phi_1\phi_2 \tag{3.35}$$

By finding a correspondence between the free energy per lattice site and the free energy per particle in DPD, Groot and Warren were able to determine the following relations between the Flory-Huggins χ parameter and the excess repulsion parameter Δa for a constant density $\rho = 3$.

$$\chi = (0.286 \pm 0.002)\Delta a(\rho = 3) \tag{3.36}$$

When creating new DPD models of polymers, there is a further step which is required. The χ parameter provides the conservative force parameter for the DPD models, but it is also necessary to determine the bond length and angle parameters such that the bonds in DPD accurately represent the bonds in the molecule. Ortiz et al. proposed a method for deriving these parameters when creating a model polyethylene oxide-polyethylene ethylene (PEO-PEE) polymer [204] which involves the simulation of the polymer in both MD and DPD. In the MD simulations, it is necessary to calculate the average centre of mass of the atoms which will compose the coarse grained DPD bead. So for example, if a bead represents one PEE monomer in DPD, then the centre of mass of the molecules of one PEE monomer would be equivalent to one DPD bead, and so it is then necessary to observe the distances and angles between the average monomer positions. Parameters can then be determined in DPD (either by trial and error or by some parameter optimisation technique) so

that the mean and standard deviation of the bond lengths and angles are reproduced correctly.

The Physical Interpretation of DPD Models

Early DPD simulations were intended to act as general models of amphiphile and fluid behaviour, and as interaction parameters were typically derived empirically, little consideration was given to how the simulated entities mapped to physical length and timescales. In [111] Groot and Rabone consider the question of how the setting of the DPD parameters relates to physical length while performing simulations of membrane damage. The simulation involves the investigation of membrane damage under increasing strain, and so a bilayer membrane composed of model amphiphiles, which were based on the model phosphatidylethanolamine (PE) amphiphiles very similar in structure to the dimyristoylphosphatidylcholine (DMPC) model amphiphiles used in the simulations presented in this thesis. The physical length scale in a DPD simulation is determined by the number of water molecules that a solvent bead represents N_m. A water molecule has a molecular volume of $30\mathring{A}^3$, and the authors chose $N_m = 3$ when creating the model PE amphiphiles. This means that each water bead represents a volume of fluid equivalent to three water molecules or $90\mathring{A}^3$, and as the bead density $\rho = 3$ a cube with side length of the force interaction distance (r_c) will contain three beads, therefore representing a volume of $270\mathring{A}^3$. By defining this volume, it is possible to calculate the approximate physical length of the force interaction radius, which is generally used to define the unit length of the simulation.

$$r_c = \sqrt[3]{270\mathring{A}^3} \tag{3.37}$$

$$= 6.4633\mathring{A} \tag{3.38}$$

The authors then determined the physical interpretation of the time unit in DPD, by matching the self diffusion constant of the water beads with that of water. The self diffusion constant of a water bead in DPD was determined from simulations to be

$$D_w = 0.1707(14)R_c^2/\tau \tag{3.39}$$

which is equated to the experimental diffusion constant of water $D_{water} = 2.43 \pm 0.01 \times 10^{-5} cm^2/s$ which gives the following length for the timestep.

$$\tau = \frac{3 \cdot 0.1707(14)R_c^2}{2.43(1) \cdot 10^{-5} cm^2/s} = 88.0 \pm 0.8ps \tag{3.40}$$

The interpretation of the unit of energy in DPD comes from the assumption that the fluid in DPD is modelled as water at room temperature. Room temperature is typically considered to be $20°C$,

and so the unit of energy in DPD using these parameters is:

$$\epsilon = 4.047 \times 10^{-21} J \tag{3.41}$$

3.3.3 DPD Simulation Software

Several publicly available implementations of the DPD software exist, although the method is typically a small component of larger simulation packages, such as the *Complex Fluid Simulations* package produced by GmbH, the *Materials Studio* from Accelrys, the *Fluidix suite* from OneZero Software and *Cuilgi*, with the open source *DL_MESO* package provided by the science and technology facilities council. However, as this study involved the routine simulation of volumes which were at the limit of what is feasible with the DPD technique in terms of computational requirements, and non standard extensions to the DPD method, such as the addition of chemical reactions were required, a new implementation of the DPD technique was created in C++, to run on multiple parallel platforms such as MPI and Nvidia's CUDA platform. Full details of this implementation are given in Chapter 4.

3.3.4 The Stochastic Simulation Algorithm

For simulation of populations of vesicles, large vesicle computing models, or simulations over long time scales, Gillespie's stochastic simulation algorithm (SSA) [104, 105] is a more appropriate choice. The vesicle computing framework supports the execution of stochastic and lattice P system models using an extended version of the SSA which allows simulation of multi-compartment structures of *stochastic P systems* [223]. SSA is a commonly used simulation technique, which is concerned only with reactions between a multiset of reactants contained within volume encapsulated by an abstract representation of a membrane, offering a good trade-off between computational efficiency and level of detail. Using SSA, a system with an initial state containing the number of molecules of each chemical species, and a set of reactions which are assigned stochastic rate constants can be simulated to produce a stochastic Markov chain, which exactly represents the distribution of the chemical master equation for the system.

The fundamental hypothesis of Gillespie's SSA is that given a system of X_i molecules of chemical species $S_i (i = 1, ..., N)$ which can react according to M well defined reactions channels $\mu_i (i = 1, ..., M)$ there exists M constants $c_\mu (\mu = 1, ..., M)$ such that the average probability that a pair of reactants will react within the given volume V in the next infinitesimal time increment dt is given by the following expression.

$$c_\mu dt \tag{3.42}$$

Therefore if the c_μ constant is multiplied by the number of combinations of reaction pairs in V then the result is the probability that the reaction μ will occur anywhere within V within the infinitesimal increment dt. The value for c_μ is derived as follows. Given two molecules in solution, and assuming

that the system is in thermal equilibrium (e.g. the composition of the volume is homogeneous and well mixed), then during an infinitesimal time increment δt one molecule will sweep out a collision volume δV_{coll} relative to the other.

$$\delta V_{coll} = \pi r_{12}^2 v_{12} \delta t \tag{3.43}$$

Where $r_{12} = r_1 + r_2$ is the sum of the two molecule's radii and v_{12} is the relative speed of the two molecules. Since the system is well mixed, the probability that the other molecule will enter that collision volume during the time increment is simply the ratio between the collision volume and the total system volume $\delta V_{coll}/V$.

$$P_{coll} = \frac{\pi r_{12}^2 \overline{v_{12}} \delta t}{V} \tag{3.44}$$

Assuming gas kinetics, the relative speeds between the two particles can be estimated by calculating the mean speed from the Maxwell-Boltzmann distribution and so the probability that a single pair of molecules will collide within the time increment δt can be derived: $\overline{v_{12}} = (8kT/\pi m_{12})^{1/2}$ where k is Boltzmann's constant, T is the system temperature and m_{12} is the reduced mass of the two molecules. The probability of a reaction occurring in the infinitesimal time period dt is given by multiplying the probability of a collision resulting in a reaction P_{react} by the probability of collision, which gives c_μ, the average probability that a pair of reactants will react within dt.

$$c_\mu = P_{coll} P_{react} dt \tag{3.45}$$

Therefore the stochastic time evolution of a chemically reacting system can be performed by determining repeatedly at what time the next reaction will occur, and which reaction it will be. The probability distribution of these values is given by the "Reaction Probability Density" function $P(\tau, \mu)$ which gives the probability that a reaction μ will occur at time $t+\tau+d\tau$, a joint probability function on the space of the variable $0 \geq \tau < \infty$ and the discrete variable $\mu(\mu = 1, 2, ..., n)$. In order to assign values to the τ and μ variables, Gillespie derives an analytical expression for the function by first considering the probability that a reaction will occur in $\tau + d\tau$ time units is the product of the probability that no reaction will occur during the time $t + \tau$

$$P_0(\tau) = \exp\left[-\sum_{v=1}^{\mu} h_v c_v \tau\right] \tag{3.46}$$

(where h_v is the number of reactant combinations in the volume at τ for reaction v, and c_v is the stochastic rate constant for v defined in Eq. 3.42.) and the probability that a reaction μ will occur in the infinitesimal time increment $\tau + d\tau$ which is:

$$P_\mu(\tau + d\tau) = h_\mu c_\mu dt \tag{3.47}$$

The reaction probability density function can then be defined

$$P(\tau, \mu) = h_\mu c_\mu exp(-a_0 \tau) \tag{3.48}$$

where a_0 is the sum $\sum_{v=1}^{M} h_v c_v$. The SSA algorithm operates by advancing the system one reaction at a time, selecting at each iteration how long to wait before the next reaction, and what the next reaction will be. Two random numbers r_1 and r_2 are generated from a uniform distribution and are then transformed to the $P(\tau, \mu)$ distribution to determine the time and type of the next reaction.

$$\tau = (1/a_0) ln(1/r_1) \tag{3.49}$$

taking μ to be the integer for which

$$\sum_{v=1}^{M-1} h_v c_v < r_2 a_0 \leq \sum_{v=1}^{M} a_v \tag{3.50}$$

The algorithm for advancing the system using Gillespie's method involves repeatedly drawing pairs of numbers from the distribution, adjusting the number of the reacted chemical species and incrementing the time and is given in Algorithm 1 [182]. The *CalcPropensity* function cal-

Algorithm 1: Pseudocode for Gillespie's "direct method" of implementation of the stochastic simulation algorithm.

```
begin
    S[1..M] ← Initial species populations
    R[1..N] ← Reactions
    TotalPropensity ← 0.0
    CurrentTime ← 0.0
    while CurrentTime < Endtime do
        for I ← 0to N do
            Prop[I] ← CalcPropensity(S,R[I])
            TotalPropensity ← TotalPropensity + Prop[I]
        end
        T ← -ln(rand())/TotalPropensity
        Selector ← TotalPropensity * rand()
        for I ← 0 to N do
            Selector ← Selector − Prop[I]
            if Selector ≤ 0 then
                SelRxn ← I
                Break
            end
        end
        S ← S − R[SelRxn].reactants + R[SelRxn].products
        CurrentTime ← CurrentTime + T
    end
end
```

culates the propensity for a reaction during the current iteration of the algorithm. The propensity is calculated from the stochastic rate constant and the number of combinations of reactant pairs in the system. For example a simple reaction μ between two different reactants X and Y with $|X|$ molecules of X and $|Y|$ molecules of Y in the system at the current iteration, the propensity p_μ is

given by Equation 3.51.

$$p_i = c_i|X||Y| \tag{3.51}$$

In the rest of this thesis, a modified version of the Gillespie algorithm proposed in [223] is used. The pseudo-code for this algorithm is given in Algorithm 2. For each compartment the standard Gillespie Direct Method [104] (`GillespieDirectMethod()` in Algorithm 2) associates the propensity to each rule associated with a compartment. A call to `GillespieDirectMethod()` for a compartment i returns the rule ρ with the shortest waiting time τ. The algorithm uses an indexed priority queue to schedule the compartments according to the waiting time of the next rule to be applied (i.e. the rule with the shortest waiting time). After the application of this rule the global simulation time t is advanced accordingly and the new rules to be applied and their waiting times are recomputed in the compartments affected by the application of the rule. Simulation continues until either a specified maximum time t_{max} is reached, or no further rules can be executed.

Algorithm 2: Multicompartment Gillespie Algorithm with Queue

```
    begin
        // preprocess
        // calculate waiting time for each compartment
 1      for i ← 1 to number of compartments do
            // perform Gillespie Direct Method
 2          (τ, ρ) ← GillespieDirectMethod(i)
            // add waiting time to queue
 3          QueueInsert(τ, i, ρ)
 4      end
        // main loop
 5      while (τ, i, ρ) ← QueuePeek() do
 6          if τ > t_max then
 7              halt
 8          end
            // advance simulation time
 9          t ← τ
            // apply rule ρ in compartment i
10          ExecuteRule(ρ,i)
            // calculate waiting time for compartment i
11          (τ, ρ) ← GillespieDirectMethod(i)
            // add waiting time to queue
12          if τ = 0 then
                // no rule applicable
13              QueueDeleteHead()
14          else
                // replace waiting time
15              QueueReplaceHead(t + τ, i, ρ)
16          end
            // update target compartment of rule
17          if rule ρ in compartment i is a translocation rule then
18              set i to index of target compartment of rule ρ
                // calculate waiting time for compartment i
19              (τ, ρ) ← GillespieDirectMethod(i)
                // add waiting time to queue
20              if τ = 0 then
                    // no rule applicable
21                  QueueDeleteEntry(i)
22              else
                    // replace waiting time
23                  QueueReplaceEntry(t + τ, i, ρ)
24              end
25          end
26      end
    end
```

The automated analysis of properties of a vesicle computing model is provided by the addition of a model checking technique to the framework. The model checking software used for this framework is described in the next section.

3.3.5 Simulative Model Checking

Model checking is a formal verification technique which allows the modeller to determine whether a model fulfils a certain set of criteria which can be specified in a rigorous manner using propositional logic. Although model checking is fairly recent development, it is widely used in industry to formally verify the correctness of specifications for systems which have a large number of possible states (processor designs or communication protocols for example) [60, 43]. Model checking has also been applied to validate models in systems and synthetic biology and in this respect, model checking has been applied to the analysis of cellular models [30, 123, 254] according to the executable biology approach of bringing into systems and synthetic biology formal and principled methodologies successfully applied in computer science and engineering.

Model checking is a highly automatic methodology that requires the user to specify the model of the system in a high-level formal specification language like Petri nets [125], process algebra [59] or P systems [211] in the case of this simulation and modelling framework. The properties or specifications to be checked against the model are then expressed using an appropriate *temporal logic*. Model checking tools take the formal model and temporal logic properties as inputs and typically convert the model automatically into a finite state transition graph and apply efficient search algorithms to either verify the properties or produce counterexamples. The main limitation of model checking consists in its high computational cost that prevents its application to large system due to the *state explosion problem*. This was addressed by the introduction of a symbolic representation for the transitions of complex systems as Boolean formulas that are then represented as *ordered binary decision diagrams* [43]. This new approach in model checking was termed *symbolic model checking* and has enabled the verification of extremely large systems with more than 10^{20} states [44].

The evident importance of noise and stochasticity in many real-life systems, like the models in this thesis, has motivated the development of a variant of symbolic model checking, termed *symbolic and probabilistic model checking* [208], able to provide quantitative information about the performance of systems with stochastic behaviour. This variant typically converts the formal specification of the system into *continuous time Markov chains* (CTMC), which allows the analysis of quantitative properties.

In this study of vesicle systems with stochastic dynamics, the properties to be analysed need to be expressed in an appropriate logical formalism capturing the characteristics of continuous time Markov chains. One such formalism in the temporal logic called Continuous Stochatic Logic (CSL) [26, 27]. This temporal logic is an extension of the branching logic CTL (Computation Tree

Logic) [80] and its probabilistic counterpart PCTL (Probabilistic Computation Tree Logic) [119], which allows the specification of real-time probabilistic properties including path-based, state-based and steady-state measures of CTMCs.

A number of different model checking software packages are available, and for model checking in this work the PRISM package was used as it combines a convenient user interface with a powerful underlying model checking engine. Two different model checking approaches are available in this software. Firstly the analytical model checking approach, which involves the full exploration of the model state space and simulative model checking, which enables the calculation of average values by simulating many different instances of the model as a set of CTMCs in the hope that a representative number of states in the model will be visited.

Recently, the possibility of using probabilistic model checking for analysis of biological models in systems biology was proposed [149]. Since the state space of biological models is very large, and the models themselves are continuous and stochastic, it is generally not possible to apply formal model checking to determine with absolute certainty whether a certain property holds for a model. Instead, PRISM enables properties to be tested against ensemble average behaviours of discrete stochastic simulations. The simulations which are performed are very similar to those based on the SSA method, except that the PRISM software enables the easy interrogation of the models via properties specified in continuous stochastic logic (CSL), and allows the user to precisely request the precision and confidence of the estimate for the value which is being considered. Stochastic P systems models can be converted automatically to PRISM model specifications in the framework using the *Multi-compartment Stochastic Simulations* program in the infobiotics workbench. The user must then specify the required precision and confidence for the estimate $\hat{p}$ of the quantity p according to the following relation:

$$P[\ |p - \hat{p}| > precision\ ] < confidence \tag{3.52}$$

Model checking in PRISM involves the specification of the model to be checked, which is constructed using the PRISM model language, and a set of properties which are expressed formally in temporal logic, using PRISM's property specification language. PRISM supports the specification and analysis of model properties based on the notion of *rewards*. The reward mechanism enables flexible accounting of a much wider range of properties than the probability of a model behaving in a certain way. For example, the specification of rewards enables the computation of properties such as the *Expected Time* for models, which is used later in this thesis for the analysis of the vesicle computing logic gates. Rewards in the PRISM model are specified in the following form:

```
rewards
    guard : reward;
endrewards
```

where the *guard* specification is a predicate for a model variable (e.g. "Protein1 > 0"), and the

reward is a numerical reward assigned to states which satisfy the predicate (e.g. "100" or "2*x"). PRISM can analyse the properties of models relating to the expected values of rewards, by specifying the appropriate query using the **R** operator, in the following form:

```
R =? [ rewardprop ]
```

PRISM allows the specification of four different reward properties. Reachability reward properties correspond to the reward accumulated along a path until a specified predicate becomes true, so for example the following property determines the accumulated reward until the variable x equals five.

```
R =? [ F x=5 ]
```

Cumulative reward properties are similar to reachability reward properties, but only accumulate the reward until a specified time t. So for example, if a reward of 1 was assigned every time there was a state transition in a model corresponding to the transcription of a gene to a messenger RNA, the following property would determine the number of transcriptions occurring before time $t = 10.0$.

```
R =? [ C<=10.0 ]
```

The instantaneous reward property can be used to determine the reward for a state at 100 time units.

```
R =? [ I=100.0 ]
```

The use of model checking in this thesis can then be seen as an aid to model construction in vesicle computing, as it enables the modeller to test model behaviour in a rigorous manner. Using PRISM involves the specification of the model in the PRISM language, a scripting language which enables the developer to fully specify the model in terms of states and rules which move the system between states. To interrogate the model, the modeller can formulate queries using a number of different temporal logics. This allows a validation of modular models of vesicle computing systems, such that behaviours (of model logic gates, for example) can be characterised with a high degree of certainty.

3.4 Summary

In this section, the modelling techniques which were used to perform the investigation of liposome logic in the rest of this thesis have been introduced. A multi-scale approach was take when considering the simulation of these systems, as the dynamics of interest occur over different length and timescales. Dissipative particle dynamics was chosen for the mesoscale simulations, as it allows a high level of detail when simulating membrane systems and is not as computationally expensive as full atomistic simulation techniques such as molecular dynamics. Also, the inclusion of momentum conservation within the technique allows a full description of the hydrodynamics aspects of the model

bilayers. It has been previously shown that DPD is a good technique for the simulation of vesicle formation.

For the longer timescale aspects of the liposome logic simulations such as the simulated gene regulatory network reactions, stochastic lattice population P Systems were chosen as they allow a formal specification of models which include interactions between colonies of bacteria or vesicles, and are based on the stochastic simulation technique, which produces a stochastic Markov chain which accurately represents the dynamics specified by the chemical master equation for the system.

Model checking techniques are also employed in the form of PRISM, the probabilistic model checker. The model checking approach enables a more formal verification of observable values within a stochastic simulation framework, and enables questions to be asked about models in a convenient and precise way using continuous stochastic logic queries, coupled with the reward system in PRISM. The model checking described in this work was used to determine properties of the model gene regulation logic gates, such as the propagation delay.

Since there are no freely available implementations of the DPD technique and non standard extensions that permit the simulation of chemical reactions within the self-assembled membranes are required, it was necessary to create a new implementation of DPD. In the next chapter, a high performance parallel implementation of DPD, intended to run distributed memory clusters and CUDA enabled graphics processing units is described in detail alongside chemical reaction extensions to the model.

Chapter 4

The Dissipative Particle Dynamics Toolkit

The simulation of local interactions between large numbers of particles for large numbers of time increments means that the DPD method is a computationally expensive technique. In order to simulate fully the reaction dynamics of vesicle computing systems, and to enable the simulation of the relevant length and time scales a high performance parallelised implementation of the DPD method was created. In this chapter the details of two high performance parallel implementations of DPD are presented, along with extensions to the DPD method to support the modelling of chemical reactions.

4.1 Introduction

The formal description of the DPD method given in the previous chapter indicates that DPD systems cannot be solved analytically, even for very small systems. Instead, numerical methods must be used to find an approximate solution to the equations by repeatedly incrementing the system time by small but finite steps and calculating the forces, velocities and positions at each timestep based on the values from the previous timestep. Although Dissipative Particle dynamics is a coarse grained method, the simulation of vesicles at the mesoscale where the amphiphiles composing the membrane are individually represented is highly computationally expensive.

The need for a highly optimised implementation of the algorithm along with the requirement for non standard extensions to the method such as chemical reactions, suggests that the existing implementations of the DPD method will not meet the requirements for the investigations of vesicle computing systems, and so as part of this work a new implementation of the DPD method was created. Implementing DPD for simulations of systems large enough to contain vesicles in a manner which makes as efficient use of the available computational hardware as possible requires careful consideration of the algorithms and data structures when designing the software.

In an effort to increase size of the simulations that can be routinely performed, it is common practice to extend a particle dynamics implementation to make use of parallel computing architectures, and so by dividing the simulation effort over multiple processors, it is possible to increase

the number of time steps of the simulation which can be simulated within a given period of time, or increase the size of the simulation. Traditionally, particle dynamics code has been parallelised for shared or distributed memory clusters, and a body of literature is available which describes efficient parallel algorithms for methods such as molecular dynamics [212, 213, 248, 219] which have since been applied to the DPD method in [241] where the authors discuss two different approaches for parallelising the DPD algorithm, previously applied in molecular dynamics simulations [267], to run on shared and distributed memory clusters. There has been increasing interest in the possibility of using the graphics processing units (GPUs) available in most desktop machines for scientific computing [205].

In graphic intensive computer applications such as games, a scene is rendered on the screen by sending a set of vertices, each describing a point in 2D or 3D space and other shading and lighting data to a dedicated GPU, which then performs manipulation on the vertex data before rasterising the data and performing post processing to produce the final image, which is then displayed on the screen. The processing of vertices is an operation which is well suited to data level parallelism, and so graphics hardware manufacturers have embraced parallel processor designs to a much greater extent than CPU manufacturers, such that a modern GPU may contain over 250 sub-processors.

Although individually these sub-processors are not as powerful as a standard CPU, in combination they provide a modern GPU with a theoretical maximum processing speed which is typically much greater than that provided by the machines CPU, approaching one teraflop/s (trillion floating point operations per second). Researchers in the field of parallel programming initially attempted to make use of these large scale parallel processors by encoding problem data as textures, and programming parallel algorithms using esoteric shader languages. More recently, graphics hardware manufacturers have enabled general purpose GPU (GPGPU) computation on GPUs by providing device driver APIs and language extensions that enable the programmer to program the GPU using C, C++ or other standard languages. Currently, there are two major GPGPU frameworks, the Compute Unified Device Architecture (CUDA) which is a toolkit, library and C/C++ compiler provided by Nvidia which allows the program to write general code to run on Nvidia graphics cards, and OpenCL, which is a proposed standard from the Chronos group, intended to provide a standard set of libraries and language extensions which will work over many different data parallel hardware implementations.

The implementation of DPD created for this work was parallelised so that it could be executed in two different parallel environments, on the University of Nottingham cluster using MPI, and using CUDA so that the simulations could be performed on Nvidia graphics cards. The University of Nottingham cluster is a large shared high performance distributed memory parallel computing facility containing 200 nodes, each with two quad core Intel Xeon E5472 processors, giving a total of 1600 cores. CUDA is a highly parallel programming environment in which lightweight computational threads run in parallel on a GPU, with little or no communication between them. DPD is a method that presents a number of unique challenges during implementation with CUDA, and so at

the time of writing, the software described below is the first CUDA based implementation of DPD. The parallel and CUDA based algorithms are described in detail in this chapter.

A further requirement of a simulation and modelling tool is the provision of functionality for analysing the data which the simulation produces. In particle simulations there is the propensity to generate huge quantities of data as positions and velocities could be recorded for every particle at every timestep. Since storing this data is often unnecessary as it is the more macroscopic properties of the system which are of interest, it is necessary to perform both "online" and "offline" analysis of the simulation data to extract the properties of the system which are of interest. For vesicle computing, the formation and dynamics of vesicle membranes are of interest, and so analysis techniques which enable the automatic identification analysis and extraction of vesicles from the simulated data are required.

This chapter details requirements and implementational details of DPD implementations running on desktop, distributed memory and GPU platforms, along with the tools for managing, analysing and altering vesicle data. The applications form a toolkit for the modelling simulation and analysis of vesicles and vesicle computing at the mesoscale. The performance of the implementations on different platforms is compared and solutions to difficulties which are particular to the CUDA implementation of the method, such as pseudo random number generation (PRNG), are described in detail.

4.2 Requirements and Analysis

To illustrate the requirements for DPD simulation when investigating vesicle computing, some possible use cases are considered. These use cases give insight into the requirements for the DPD software.

- Use Case 1: The modeller describes a vesicle formation simulation as a configuration file, and then performs the simulation with the DPD simulator software. The results of the simulation might include statistics calculated over the course of the simulation, or data about particle positions and velocities. This data will be written to an intermediate file, which is then read by a display and analysis application which displays the state of the 3D volume at each recorded timestep.

- Use Case 2: Data recorded from the DPD simulation can be loaded into the display and analysis software, and analysed to detect self-assembled objects of interest (e.g. vesicles and bilayers). These objects can then be extracted and stored as separate object data files.

- Use Case 3: A new initial state for a vesicle computing simulation is created by placing self-assembled objects within a 3D volume using the initial state design software to represent the structure of a proposed vesicle computation. This new state is used as input to the DPD simulator, which then evolves the system from that state.

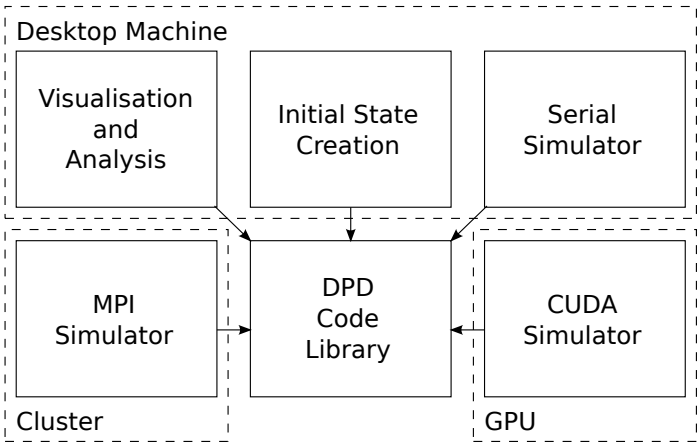


Figure 4.1: Schematic representation of the DPD toolkit, showing the relationships between the different tools. Tools are contained within dashed boxes that indicate on which platform the tool will run. Arrows indicate a code dependency between the tools (e.g. all components use code in the DPD shared code library, which contains classes representing Coordinates, particles etc.)

- Use Case 4: A new initial state for a vesicle computing simulation is created from pre-simulated self-assembled object, as in Use case 3. However, in this case, the modeller modifies the contents of the vesicle(s) to include vesicle computing molecules. The volume is then saved as a data file which can act as the input to the DPD simulator.

As well as fulfilling the use case requirements listed above, the performance of the DPD simulator application will be of great importance, as integrating the equations of motion for a large number of particles is a highly computationally expensive operation. The choice of algorithms is of key importance in ensuring that the simulations can be performed in the required period of time.

The DPD simulation evolves the position and velocities of millions of particles over millions of timesteps, and so the simulator is capable of producing large amounts of data. Since the DPD simulator is designed to run for long periods of time unattended, the visualisation and analysis of data must be separated from the data generation process, and so the DPD toolkit is built around the concept of data file generation using the DPD simulator, so that the data file can be stored for later analysis. A number of different programs were created to visualised, manage and analyse the data resulting from DPD simulations, and these programs along with a shared library make up the DPD toolkit. A schematic diagram of the DPD simulation toolkit is given in Figure 4.1, showing the different tools and the platforms on which the tools are intended to run. The different applications in the DPD software toolkit are now described in more detail.

4.3 The DPD Simulator

The DPD simulator is used to perform the simulation of DPD systems. Since the performance of the DPD simulation code is critical, rather than attempting to create one large application which

can be executed on the three targeted platforms (a single CPU, a distributed memory cluster, and a CUDA enabled GPU) three separate programs were created, a serial version of the code designed to run on a single CPU, a parallel version of the code implemented using MPI and designed to run on traditional distributed memory clusters, and a highly parallel CUDA version of the code which runs on Nvidia GPUs.

By creating three different programs, it was possible to optimise each version of the DPD simulator specifically for its target platform, without requiring a compromise to be made between generality (e.g. code that is general enough to support all three platforms) and performance. However to reduce the development time and make maintenance easier the non performance critical sections of the code which were common amongst all three versions of the software (such as the configuration classes) were stored in a shared library.

4.3.1 A Serial implementation of DPD

The serial implementation of the DPD method is designed to run on a single processor, but many of the data structures and techniques applied to this implementation of the method are used in the parallel and CUDA versions of the code. Since the implementation of the DPD method is essentially a numerical integration of the particle positions and velocities, standard numerical integration techniques for particle dynamics methods can be considered when implementing DPD. The available integration schemes which could be applied to calculate forces in DPD are now considered.

Integration Schemes

The choice of integration scheme in a particle dynamics simulation is important, as a good integration scheme will permit longer time steps to be taken and therefore reduce the CPU time required for a simulation. The integration of the DPD algorithm is more complicated for DPD than it is for MD, as the calculation of the dissipative force (Eq. 3.11) requires the relative instantaneous velocity of the particle pair, and the calculation of the velocity requires the dissipative force value. The choice of integration scheme requires a trade-off to be made between the following criteria:

- Time reversibility - If a system is integrated and then the integration algorithm is applied in reverse, the system should return to its initial state along the same trajectory through phase space.
- Accuracy - The integration scheme should minimise the numerical error.
- Computational Efficiency - The integration scheme should not be too expensive to calculate.
- Space Efficiency - The scheme should minimise the amount of particle data required to perform the integration.

Regardless of which numerical integration method is used there will be a small error between the approximate value and the actual value of the function, and these errors will accumulate over the duration of the simulation. The magnitude of the error at each step is a function of the integration step size, and so the time step size is a common parameter in particle dynamics simulations which allows the modeller to trade accuracy for performance (i.e. a longer time step will introduce a greater function approximation error than a shorter one, but a larger amount of time can be simulated for the same computational effort).

A number of different integration schemes for DPD have been proposed, ranging from extensions to the standard MD velocity verlet integration scheme [206, 37] which take into account the force-velocity dependency by recalculating the dissipative force at the end of each integration step or calculating an estimated velocity [110], to exotic methods such as the Otter-Clarke integrator [69] which is an Euler style integrator in which the contributions from the conservative and dissipative forces are scaled by predetermined parameters which must be determined a-priori from simulations, and the Lowe-Anderson method [166] which no longer requires the recalculation of the random and dissipative forces between each particle pair, instead drawing values for the relative velocities between particles from a heat-bath distribution. A number of review articles have been published on the topic of DPD integration schemes, and in depth comparisons can be found in [192, 257, 37].

The integration scheme in the implementation of DPD presented in this work was chosen to satisfy the competing goals of performance and accuracy. As the CUDA version of the DPD code does not implement verlet lists, those integrators which required a secondary calculation of the dissipative force at the integration step were ruled out, as without verlet lists this recalculation would require a second traversal of the cell tables to find particle pairs which were within each others force interaction radius, which would essentially double the force calculation time. The best of the remaining candidates were the Lowe and Groot Warren integrators, as these do not require a recalculation of the dissipative force, or time consuming pre-simulations to determine integrator parameters like those required for the Otter and Clarke integrator. Of these two, the Groot Warren integrator was chosen, as it is the most widely used integrator in the DPD literature, and was among the best performing integrators in the studies listed above.

The Groot Warren integrator requires a coefficient λ to be specified prior to simulations, which is added to the intermediate calculation of the velocity for the forces. If the λ parameter is set to 0.5, then the algorithm is equivalent to the molecular dynamics velocity verlet algorithm. By setting the λ force to values greater than 0.5, the magnitude of the predicted velocity vector for each particle is increased, meaning that the magnitude of the dissipative force will be effectively increased during the force calculation. Typical choices for the λ parameter are 0.5 or 0.65, although there is no theoretical support for setting this parameter, and so it should be tuned empirically. The λ parameter was set to 0.65, and the timestep length to 0.05 in the simulations unless otherwise specified. Algorithm 3 shows the Groot and Warren integration algorithm. Note that the dissipative force is calculated with the estimated velocity v_2.

Algorithm 3: The Groot Warren DPD integration algorithm. Each iteration, the velocity $v2$ and predicted velocity $v1$ is calculated from the forces calculated in the previous iteration, the particle positions *pos* are then updated, the cell tables (*tables*) rebuilt and then the forces are calculated using the cell tables method for each particle i.

```
for i ← 1 to n do
    v2 ←v1 + calculateVelocity(dt = λ ∗ dt,forces);
    v1 ←v1 + calculateVelocity(dt = 0.5 ∗ dt,forces);
    pos ←pos + calculatePositions(pos,v1,dt);
    tables ←createCellTables(pos);
    forces ←calculateForces(tables,v2,dt);
    v1 ←v1 + calculateVelocity(dt = 0.5 ∗ dt);
```

Using this integrator and the parameters $\sigma = 3.0$ and $\gamma = 4.5$ for the random and dissipative force the system temperature remained very close to the theoretical value of $1k_bT$ for the conservative force parameter $a = 25k_bT$, although it is interesting to note that for systems with an increased bead number (the number of molecules represented by each bead) of 3, with a corresponding $a = 78k_bT$, the average system temperature dropped to $0.985k_bT$. This drop in temperature does not seem significant enough to alter the conclusions presented later on, and could be rectified by slightly decreasing the λ parameter.

The calculation of the inter-particle forces is the most computationally expensive section of the above integration loop as it is necessary to calculate distances between a particle and all of the other particles which may be within the force interaction radius to find out if they are within the force calculation range. Also, for long range forces, the number of particles which may be in the interaction radius for each particle may be very large, and so the number of force calculations is also large. The naive approach to calculating the force acting on each particle is to calculate the distance between the particle and every other particle in the simulation, which may be necessary if the force interaction radius is larger than the simulated volume. However, as the forces in DPD have a short interaction range relative to the size of the simulated volumes the force calculation can be optimised by using appropriate data structures. In systems like DPD where the density in a simulated system does not vary much over the volume, the particles can be assumed to be distributed fairly uniformly throughout the space. In such cases, the cell tables data structure can be employed to aid the efficient calculation of the inter-particle forces.

Cell Tables

Cell tables can be employed in many different particle dynamics simulations to reduce the complexity of the force calculations by binning the particles in the simulated volume into cells based on their positions [13, 219]. Each cell then holds the particles which are contained within a cubic subvolume with side lengths which are equal to the force interaction radius. The forces acting on a particle can then be calculated by determining which bin the particle belongs in, and checking the distance

between the particle and the particles contained in adjacent cells. When using cell tables in DPD, the simulation space is subdivided into cubic cells of unit volume (see figure 4.2). Each cell has

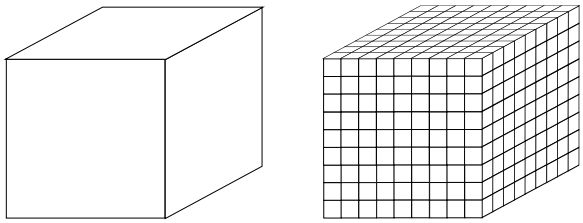


Figure 4.2: The cubic volume (left) is subdivided into smaller cubes of unit volume (right)

associated with it a list of particles that are present within the cell and will contain on average ρ particles (where ρ is the number density). Using this method, force calculation for a particle can be performed by determining which cell the particle is contained in, and searching only the neighbouring cells for interacting particles, rather than examining every other particle in the system. Once the cell tables have been constructed ($O(n)$ complexity), summing the forces acting between a particle and its neighbours involves examining the other particles within the same cell, and the particles within the surrounding $3^d - 1$ cells (where d is the number of dimensions in the simulation), reducing the complexity of the force calculation from $O(n^2)$ to $O(n)$. The construction of the cell lists involves calculating the cell coordinate for each particle, and then assigning the particle to the cell indicated by the coordinate.

Figure 4.3 illustrates the force calculation process with cell tables in two dimensions, although it is straight forward to extend the technique to three dimensions. The figure shows a further optimisation which can be achieved due to the symmetrical nature of the forces acting between each particle pair, by updating the force acting on both particle i and j at the same time, it is only becomes necessary to perform the distance calculation between particles in the same cell as the particle in question, and 13 of the surrounding 26 cells, as the force contribution between the particle and the particles in the remaining cells will be added when the forces acting on those other particles are calculated.

4.3.2 The Parallel Implementation of DPD

A simulation of the formation of a very small vesicle within a volume containing 81,000 particles for 200,000 time steps typically requires roughly 21 hours of CPU time using the serial version of the DPD software. In order for the vesicle computing simulation framework to be useful for rapid prototyping of vesicle designs, and to increase the time and length scales that can feasibly be simulated with the dissipative particle dynamics method, a parallel implementation of the code has been created to run on the University of Nottingham high performance computing cluster facility.

Larger DPD simulations can be performed by parallelising the DPD algorithm such that the computation of the particle forces are split over multiple processors. In [241] the authors investigate

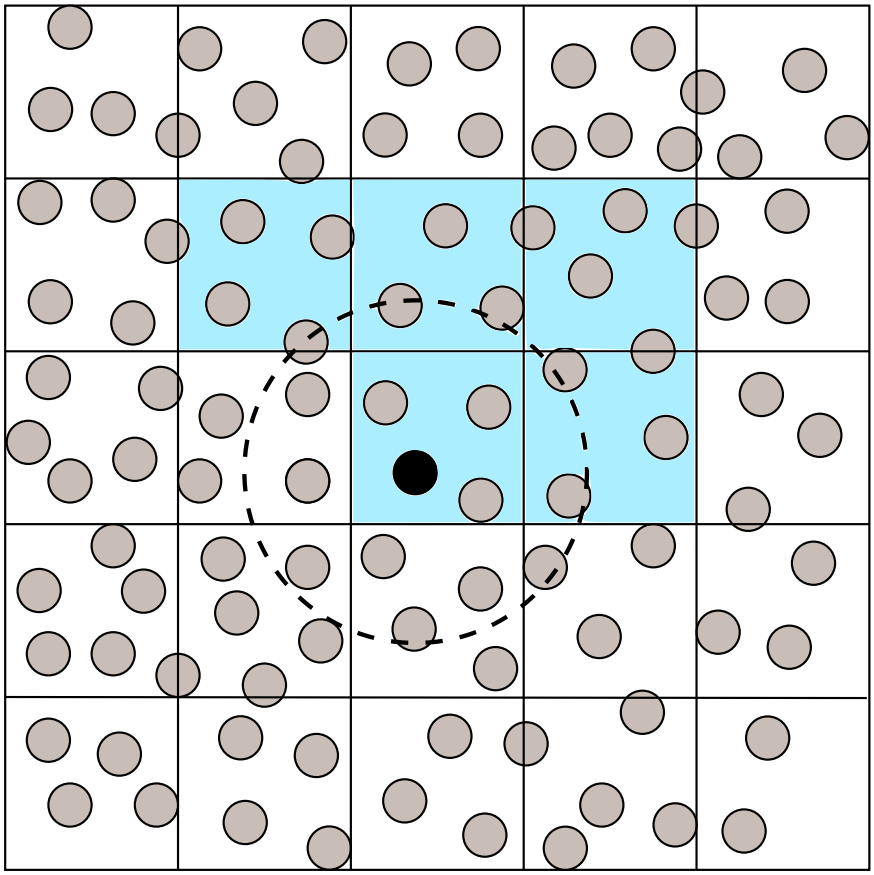


Figure 4.3: Calculating the forces acting on a particle with cell tables, shown in 2D for clarity. The grid squares correspond to cells in the cell table, and so any particles positioned within a grid square will belong to the corresponding cell. To calculate the force acting on the black coloured particle, a summation must be performed of the component forces between the black particle and the grey particles that are within the force interaction radius (denoted by the dashed circle), which are located by calculating the distance between the black particle and those particles in the neighbouring cells (coloured blue). Since Newton's third law states that $F_{ij} = -F_{ji}$, it is only necessary to sum the forces for particles within the same cell as the black particle, and half of the neighbouring cells, this is because when the forces are calculated for particles in the neighbouring cells not searched, the search will include the cell which contains the black particle.

several different algorithms for dividing the computational effort between the different processors. The methods differed in the *decomposition* of the particle data over the nodes, the manner in which the particle data was distributed between the nodes, during each time step.

For example, one approach to the parallelisation of DPD simulations which was considered by the authors was that of *replicated data* which involves maintaining a copy of all the particle data on each parallel node, allowing each node to perform an expensive operation on a subset of the data. Each processor then broadcasts the updated data to all other processors, and receives updated data from all other nodes such that all processors update their copies of the data with the modifications. The second approach investigated was *spatial decomposition*, where the simulation space is subdivided into smaller volumes, and each processor is responsible for performing calculations for particles which are within a sub-volume of the simulation space. Some communication of particle data between processors is still required, as it is necessary for a processor to transmit data regarding the particles at the boundary of its assigned sub-volume to processors calculating forces in adjacent sub-volumes. However, the amount of communication required is greatly reduced. The authors show that for both shared and distributed memory systems, spatial decomposition was more efficient and performed better than replicated data, and so this method was used to implement the parallel DPD code for the simulations performed in this thesis.

Spatial decomposition involves partitioning the simulation space along each axis, and assigning each subspace to a single processor in the cluster. If there are m processors and N particles, then a single processor only has to calculate the particle trajectories for $\frac{N}{m}$ particles. When particles cross the boundary between two processors, all particle information is transmitted between the two processors. At the processor space boundaries, it is necessary to transmit copies of particles that are within r_c distance from the boundary to the other processor. So that particles in the adjacent processor have the necessary information to calculate the particle forces, these particles are stored for the duration of the time-step and form an extra layer around the space owned by the processor. Due to their transient nature, they are termed *ghost* particles. Once forces have been calculated between all the particles owned by a processor, care must be taken to ensure that the random force calculations are correct across processor boundaries, as the random number used to calculate the force should be the same for each processor and so another transmission is required. The sequence of steps performed by one node to calculate one time-step in the parallel implementation is as follows:

The integration of a single timestep of a DPD simulation in parallel on multiple processors requires the communication of several different sets of particle data. After the position of the particles has been integrated by the *calculatePositions* method, some particles may now have position coordinates which locate them within the space managed by another processor adjacent to the one performing the position integration. The *sendRecvParticles* method determines which particles have moved outside of the processor's assigned space, and transmits them to the relevant adjacent processors, and receives data regarding the particles moving from adjacent processors into its space. It is necessary to transmit particle state information such as the position, velocity, predicted velocity

Algorithm 4: The Groot Warren DPD Integration algorithm, modified to support spatial decomposition based parallelism. Three new methods are introduced which perform communication of particle data between adjacent processors.

```
for i ← 1 to n do
    v2 ←v1 + calculateVelocity(dt = λ ∗ dt,forces);
    v1 ←v1 + calculateVelocity(dt = 0.5 ∗ dt,forces);
    pos ←pos + calculatePositions(pos,v1,dt);
    SendRecvParticles();
    SendRecvGhostParticles();
    tables ←createCellTables(pos);
    forces ←calculateForces(tables,v2,dt);
    SendRecvRandomForceUpdates();
    v1 ←v1 + calculateVelocity(dt = 0.5 ∗ dt);
```

and type.

Although transmission of particle data for particles moving across processor boundaries ensures that each processors only calculates the forces for particles located within the space assigned to the processor, further information is required to perform the full force calculation for the particles which are on the edge of the assigned space. To calculate the forces acting on these particles, it is necessary to transmit particle data for those particles which are located within the force interaction radius of the processors assigned space. The *sendRecvGhostParticles* method performs the sending and receiving of this ghost particle data, which represents a layer of particles surrounding the processors space, with a thickness given by the force interaction radius. Having accurate information regarding the particles surrounding the processors space enables the correct calculation of the forces acting on a particle at the edge of a boundary, but decreases the speed at which the computation can be performed, as each processor must perform a synchronised transmission of the particle data at each timestep.

A further communication between processors is required once the forces acting between particle have been calculated, and is due to the theoretical requirement of the DPD method that the random force calculated between two particles should be symmetrical. If a particle is at the edge of a processors assigned space, then the calculation of the forces acting on that particles requires the calculation of the force acting between that particle and one or more ghost particles belonging to an adjacent processor.

To illustrate this problem, consider two particles which are within each others' force interaction radius, but belong to two different processors, p_i^a, particle i which is owned by processor a, and p_j^b, particle j which is owned by processor b. Processor a will calculate the force acting on its particle i including the random force contribution from particle j, which involves the generation of a pseudo random number and processor b will concurrently calculate the forces acting on its particle j, including the random force contribution from particle i, generating its own pseudo random number

The problem with the above calculation is that the two processors cannot be made to

generate the same pseudo random number for the calculation of the random force acting between the i-j particle pair, as they will be using different streams of numbers from the parallel pseudo random number generator. Even using the same stream of random numbers for each processor will not alleviate the problem, as the ordering of the force calculations will be different on the different processor. Instead, it is necessary to transmit the random force contributions calculated by the processor to the adjacent processor which owns the ghost particle. For a given particle pair, whether the processor calculates the random force between the pair and then transmits it, or does not calculate the random force and instead receives it, is determined based on the indices of the two particles. For a force calculation between a pair of particles across a processor boundary, if the processor owns (i.e. has within its assigned space) the particle with the higher index, then it performs the calculation and transmits the result to the other processor. If not then the processor receives the random force contribution from the other processor and adds it to the force acting on the particle with the smaller index, Figure 4.4 illustrates this random force calculation scheme.

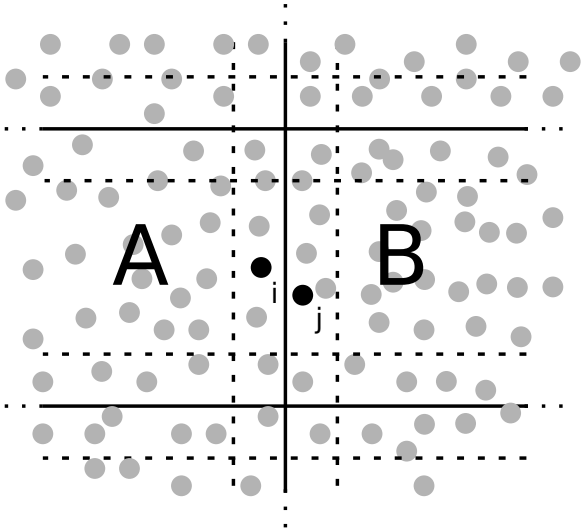


Figure 4.4: The random force problem occurs when calculating forces between particles on two different processors. The figure shows two particles, i owned by processor A and j owned by processor B, which are at the boundaries of the space owned by the processor (shown by the sold black lines). The random force F_{ij}^R calculated by processor A will not equal $-F_{ij}^R$ as the pseudo random numbers generated by each processor will be different.

To perform the transmission and reception of the particles the communication scheme outlined in the paper by Sims and Martis [241] and illustrated in Figure 4.5 is employed. The figure shows the interconnections between nodes in the cluster (a 2D simulation is chosen for clarity). The dashed arrows indicate a periodic boundary, e.g. a connection to the node on the other side of the cluster. With this scheme, for a 3D simulation the communication of particles between a processor and its surrounding 26 processors can be performed in just 6 steps. The processor sends particles to the left whilst simultaneously receiving particles from the right, and then sends particles to the

right whilst simultaneously receiving particles from the left. The same method is then employed in the Y and Z axis respectively. After each transmission, the received particles are added to the list of particles that are considered when deciding which particles to send in the next transmission. This means that it is not necessary to communicate particles directly to the processors that are diagonal to the transmitting processor.

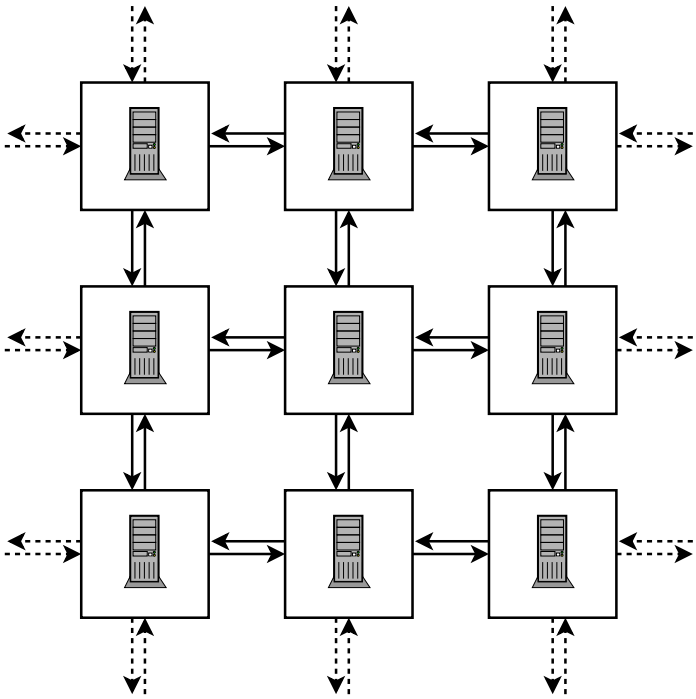


Figure 4.5: Communication between nodes in the cluster, shown in 2D for clarity. Each node sends and receives ghost particle data to the left and right adjacent nodes (The dashed arrows indicate that the boundary for this data transfer is periodic), collating the received data with the ghost particle data which is to be sent to the adjacent processors managing the space above and below the processor. This data is then sent to the above and below processor. In 3D a further collation and send/receive step is required to send particle data in the forward and backward directions. Each processor performs only 4 sends (2D) or 6 transfers (3D) rather than 9/26 sends.

Figure 4.6 shows the process of the transmission of ghost particles in two dimensions, in which only 4 communications are required. The Sub-figure 4.6a shows the communication to/from processors adjacent to the centre processor on the X axis. Particles that are within the force interaction radius, and will be sent to another processor are coloured in black, and particles that are received are coloured in grey. Sub-figure 4.6b shows the communication between the processor and processors that are adjacent on the Y axis, using the same colour scheme. Notice that particles that were received in the previous step are now being transmitted. As both processors need information about these particles, a copy of the particle information is transmitted and the original retained for force calculation.

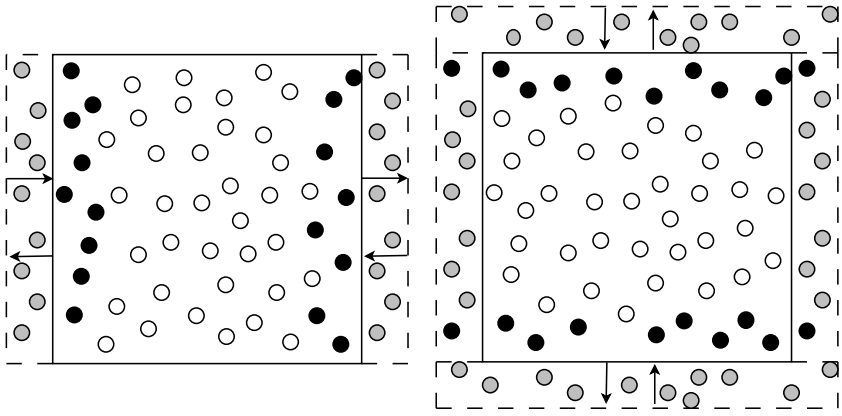


Figure 4.6: Parallel transmission of particles

Polymers In Parallel DPD

Calculating polymer forces is complex in a parallel environment, as a single cluster node may not have information about all the beads belonging to the polymer. The problem is illustrated in Figure 4.7, in which the polymer particles are coloured green and the lines indicated bonds between them. The polymer is spread over 4 processors, and so no single processor has the necessary particle information to calculate the bond forces.

This problem is overcome by extending the radius of the ghost particle perimeter for polymer particles. For a polymer which has harmonic angle potentials specified between particles. This radius should be large enough so that each processor will always have enough information to calculate the the bond angle and was determined to be $2r_c$ for the parameters used in the simulations. The downside of this approach is that there will be unnecessary communication of particle data, which reduces the efficiency of the computation.

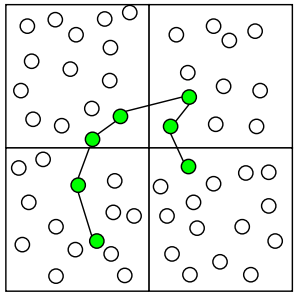


Figure 4.7: Polymer crossing processor boundaries in a parallel simulation

Using the parallel implementation of DPD on the University cluster has enabled a 200,000 time step simulation of 2187000 particles to be performed within ∼64 hours running on 27 processors, whereas a similar simulation in serial code would take over 24 days, a parallel efficiency of roughly

33 percent. This low efficiency could be explained by the data transfer overheads associated with correctly calculating the polymer forces accross processor boundaries and the relatively high latency of the 1Gbps ethernet interconnect between nodes.

4.3.3 The CUDA Implementation of DPD

The implementation of the DPD simulator which was created to run on GPUs is now described, this version of the simulator is based on the CUDA framework, which provides an environment within which programs can be written to run on the GPU hardware in C and C++.

The Compute Unified Device Architecture (CUDA)

In order to discuss the implementation of DPD within the CUDA framework, it is necessary to describe the CUDA hardware and programming model in more detail. CUDA implements a finely grained thread level parallel programming model, and threads are executed concurrently by the processors on the GPU. A CUDA GPU contains a number of streaming multiprocessor (SM) units, and each SM consists of eight scalar processor cores (henceforth referred to as cores). Thread scheduling is performed by each SM, which assigns threads to its cores for execution. The CUDA programming model has at its core three key abstractions:

- A hierarchical set of thread groups, allowing code to scale automatically to make use of new hardware.
- Shared memory, threads can read and write to shared memory which is accessible by other threads.
- Barrier synchronisation, the only thread communication primitive available in CUDA.

The thread hierarchy in CUDA is designed to be orthogonal to the hardware design, and when a parallel function is executed on the CUDA GPU, a grid is created for execution on the GPU, which contains a number of thread blocks, which each contain a number of threads. Thread blocks are scheduled to SM units, which in turn schedule the threads in the thread block to the cores in multiples of 32 threads known as "warps". Each SM unit will schedule warps of threads until all of the threads in the assigned block have executed and thread blocks will be scheduled to SM units until all thread blocks have been executed.

Algorithms which execute on the GPU are specified as "kernels", written in an extended version of the C programming language. When a kernel is executed, two parameters must be passed to the GPU which specify the number of threads in each block, and the number of thread blocks to schedule. The code in a kernel is executed concurrently by every thread in the system, and each thread has a unique thread index, which can be used to differentiate the behaviour of the running threads (e.g. the thread index might be the input into a function which calculates an index into

an array containing the data which is to be processed). A schematic layout of multiprocessors and memory on the graphics card is shown in Figure 4.8.

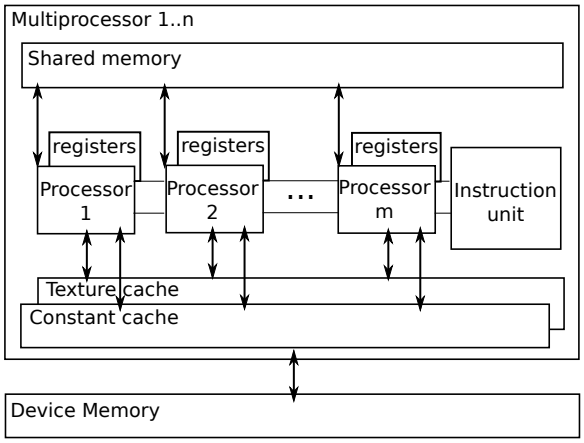


Figure 4.8: The CUDA hardware model

Three different types of memory are accessible by running threads:

- The Device (or Global) memory, the largest area of memory available in a kernel, this memory has the highest access latency and cannot be used for inter-thread communication.
- The shared memory, each SM unit has a shared memory which can be accessed by threads running on the SM unit's cores, reliable communication between threads can be achieved by writing to the shared memory and then performing a barrier synchronisation to ensure that the write has succeeded before the other thread reads the data. Note that communication between threads scheduled to different SM units is not possible.
- The register file, each core has a number of registers which have zero access latency and are used to local kernel variables.

Threads within thread blocks can communicate via a shared memory, but access must be synchronised within the thread block to ensure data consistency, which introduces a performance overhead. All threads can access the global device memory, but no synchronisation is available and it is not guaranteed that a write operation occurring in one thread will take place before the next read operation of the same location from another thread, and so the global memory should not be used for thread communication within the same kernel, a restriction which has ramifications when attempting to implement DPD in CUDA as described in the next section. The challenge in implementing an algorithm in CUDA is then parallelising the code and structuring the data in a way that fits well within the CUDA thread and memory model.

Pseudo Random Number Generation

In the CUDA DPD algorithm, each thread is required to generate one pseudo random number for each particle pair in order to calculate the random force. The threads in CUDA are very lightweight and do not have enough registers to contain the state of complicated pseudo random number generators such as Mersenne Twister (although implementations in CUDA do exist, the number of parallel streams of numbers is limited) and so instead a multiply with carry linear congruential generator first proposed by Marsaglia [179] was implemented. The PRNG has a period of around 2^{60} and requires that only 3 integers of state be stored per thread, PRNs are generated by calculating the sequence

$$x_n = ax_{n-1} + c \bmod 2^{32} \tag{4.1}$$

where x_n is the next number in the sequence, x_{n-1} is the previous number in the sequence and c is the carry from the multiplication in the previous call to the PRNG. The choice of the a multiplier is important, and should be calculated such that $a(m/2-1)$ is a safe prime. A pre-computed table of safe-prime values for a, made by extending tables provided by S. Gratton [1, 180] is loaded into memory before each run and the initial x_{n-1} and c values are chosen during the initialisation of the algorithm using serial Mersenne Twister and compose the seed for the PRNGs.

As each thread will produce a sequence of uniformly distributed PRNs, the combined sequence of PRNs should be uniformly distributed. However, there is no theoretical support for using multiple parallel PRNGs in this way, and so we resort to empirical observations to determine if the results of simulation are the same as in the serial algorithms (which use the Mersenne Twister PRNG). If this is the case, then the application of the random and dissipative forces should result in the correct reproduction of the relevant Maxwell Boltzmann distribution for particle speeds. Figure 4.9 shows the distribution of particle speeds as a probability density function overlaid with the speed probability density function calculated from the Maxwell Boltzmann distribution formula for the average particle kinetic energy. The figure shows that the mean probability calculated from DPD simulation was very close to the Maxwell Boltzmann distribution (the RMSE between the two curves is 0.0055), although not identical as the curve from the DPD experiments indicates that the average particle velocities are very slightly reduced. It is possible that the reason for this slight difference is due to the setting of the λ paramater to 0.65 reducing the overall system temperature slightly, as mentioned previously.

4.3.4 Implementing DPD in CUDA

A collision detection code [108] which utilised the cell tables approach is supplied in the CUDA SDK, and so this was used as a basis for the DPD cell tables implementation. In this example code, collisions between particles are calculated by assigning a thread to each particle in the system with each thread identifying collisions between its assigned particle and the particles in the surrounding

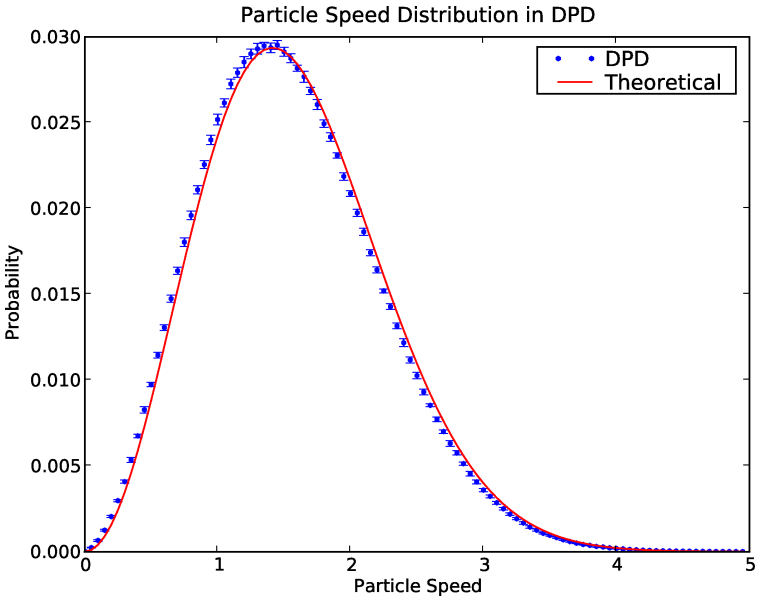


Figure 4.9: Calculated distribution of speeds from particles in a 27000 particle simulation. The blue line is the distribution of particle speeds averaged over 20 timesteps, and the red dashed line is the distribution calculated from the maxwell Boltzmann speed distribution function.

neighbourhood using the cell tables to locate neighbours. This assignment of a single thread to each particle is not appropriate for force calculation in DPD, as pairwise calculation of the random force requires the generation of a pseudo random number (PRN) for each particle pair (See random force equation 3.12). If each thread was assigned a single particle, it would be necessary to determine a scheme in which a thread, calculating the random force F_{ij}^R generated the same PRN as the thread calculating the force in the opposite direction F_{ji}^R, as PRNs must be generated by each thread in CUDA rather than globally.

In the DPD algorithm, this problem is overcome by assigning the particles contained within an entire cell to each thread. The forces between particles in the same cell are calculated by the thread owning the cell, so the symmetric property of the random force can be maintained as only one thread (with one PRNG) calculates the forces for each pair. The forces between particles that are in neighbouring cells (i.e. not in the same cell as one another) are calculated differently, by assigning the particles from a cell and one of its 26 adjacent cells to each thread. Each thread then calculates the forces between the particles in both cells, and so the symmetric random force property can be maintained. As each cell should be assigned to only one thread during each kernel call, a thread assignment algorithm is required to assign threads to cells in the cell table during each stage of the algorithm to calculate the forces between the particles in each cell and the particles in the cell's 26 neighbours. This thread assignment is performed by breaking up the force calculation into 26 separate kernel calls, and is described in detail below.

Thread Assignment

The assignment of threads in each of the 26 stages of the neighbour force calculation is performed as follows. An interleaving of the cell tables is performed along a single axis and threads are then assigned to alternating layers, so that the space is interleaved into layers of assigned and unassigned cells. Each thread calculated forces between particles in its assigned cell, and particles in an adjacent cell in the unassigned layer. During the next kernel call the index of the assigned and unassigned layers are shifted along the axis, so that the previously unassigned layer of cells is now assigned threads, and the previously assigned layer is now not assigned threads. Again, forces between particles in assigned and adjacent unassigned cells are calculated, resulting in the computation of forces between particles within each cell and two neighbouring cells (i.e. the left and right neighbours). The whole process is repeated between cells with threads assigned and the 26 adjacent unassigned cells.

Figure 4.10 illustrates this aspect of the proposed algorithm, (named "Cell Shifting") for the calculation between particles in each cell and the left and right adjacent cells. The grey coloured cells are those which are assigned threads, and the arrows indicate the adjacent cell which will be accessed by the thread assigned to the grey cell from which the arrow originated. The righthand image in the same figure shows the second stage in which the assigned and unassigned cells (grey and white respectively) are shifted, such that each thread now calculates the forces acting between the grey cells (which were previously white) and the white cells (which were previously grey), the dashed arrows indicated that the assignment is periodic in that dimension. In this way forces are calculated between each cell and the left and right neighbouring cells. Algorithm 5 shows the pseudo-code of the cell shifting method.

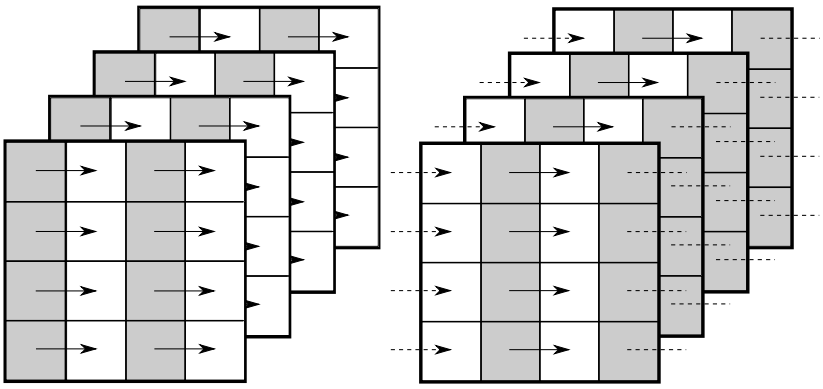


Figure 4.10: Cell shifting in the force calculation algorithm, the image on the left shows the thread assignment after the interleaving process has been applied. Cells which have threads assigned to them are coloured grey, and those without threads assigned are coloured white. An arrow pointing from a cell which has been assigned a thread indicates that the thread will calculate forces between the particles in its cell and the adjacent cell pointed to by the arrow, The figure on the right shows the assignment of threads to cells after the cell shifting takes place, with the dashed arrows indicating that the space is periodic.

Algorithm 5: The cell shifting loop, the algorithm calls the force calculation kernel for each of the 26 adjacent cells, the *calcSameCellForces*() kernel calculated the forces acting between particles within the same cell

```
calcSameCellForces();
for shift ← 0 to 1 do
    for y ← −1 to 1 do
        for z ← −1 to 1 do
            calcAdjForces(shift, idim=x, adj=(1,y,z));
    for z ← −1 to 1 do
        calcAdjForces(shift, idim=y, adj=(0,1,z));
    calcAdjForces(shift, idim=z, adj=(0,0,1));
```

As each thread is assigned particles for two cells during each computation, rather than 26, it is necessary to iterate the process 13 times to complete the search of the 26 neighbours surrounding each cell (note that during each iteration forces are calculated between each assigned cell, and only one of its neighbour cells). When interleaving the thread assignments in this way, certain neighbouring cells are not reachable by a thread, as they are within the layer of assigned cells. It is then necessary to change the plane along which the interleaving occurs, to enable computation between each cell and its neighbours.

For example, if the interleaving is applied in the z dimension as shown in Figure 4.10. Then this enables calculation of forces between particles in each cell and particles within 18 of the 26 neighbouring cells. Forces for the other 8 cells cannot be updated, because updating forces for the adjacent cell would mean altering global memory owned by another thread (in other words, the arrow indicating which neighbour is being calculated by the thread would point to another grey cell). In order to calculate the forces between each cell and the remaining eight neighbour cells, the cell interleaving process must be applied in the y and z dimensions. Figure 4.11 shows how the remaining cells can be accessed when the interleaving procedure is performed in each different dimension.

By decomposing the force calculation in this way, it is possible to perform the force calculation without inter-thread communication, whilst still conforming to the symmetric random force application requirements of the method. Algorithm 6 shows the pseudo-code for the kernel executed for each cell.

4.3.5 Performance

The performance of the CUDA DPD implementation was investigated by simulating increasing cubic volumes for 1000 timesteps with each implementation. The CUDA simulations were performed on an NVIDIA Tesla c1060, and the parallel version of the code is a self-developed distributed memory version, based on the spatial decomposition algorithm described by Sims and Martys [241] and running on the University of Nottingham High Performance Computing Facility, containing 512

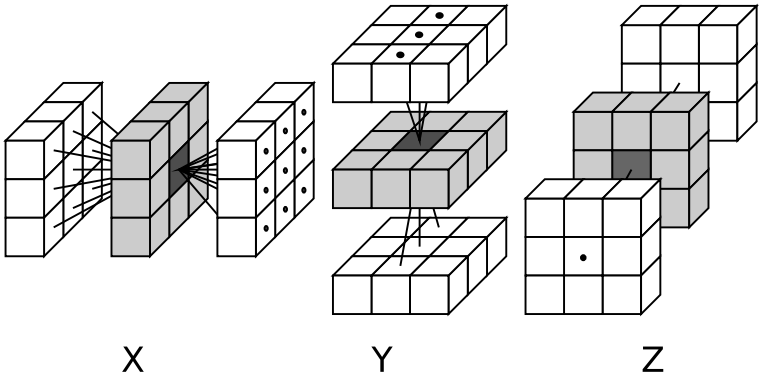


Figure 4.11: Diagram showing which neighbour cells can be accessed by the thread assigned to the centre cell (shown in dark grey) when interleaving the space in the x dimension (left), the y dimension (centre) and the z dimension (right). The light grey cells are those neighbour cells which cannot be written to by the thread assigned to the dark grey cell, as threads are assigned to them.

Algorithm 6: calcAdjacentCellForces

```
(tx,ty,tz)← calcCellIndex(threadid,interleave);
(tax,tay,taz)← calcAdjCellIndex(tx,ty,tz,shift);
celllist ← getCellList(tx,ty,tz);
adjacentlist ← getCellList(tax,tay,taz);
for pa Input: celllist
do
    for pb Input: adjacentlist
    do
        calcForce(pa,pb);
```

nodes, each with 2 AMD 2.2Ghz opteron processors. Each data point is averaged over three runs, and the performance of the parallel distributed code was run with 1,9,27 and 64 processors (as dividing the space into these numbers of processors produces cubic spatial configurations which are the most efficient in terms of the required inter-processor communication). Figure 4.12 shows the speedup the parallel and CUDA versions of the code, and indicates that the performance of the CUDA DPD implementation is faster than the parallel distributed version running on 64 nodes on a distributed memory cluster, and is over fifty times faster than the single processor version for volumes greater than $50r_c^3$.

Scalability analysis of each method in the CUDA based DPD implementation was performed using the profiling tools provided with the CUDA framework, Figure 4.13 shows the runtime for each function on the GPU and CPU respectively, for simulations of cubic spaces with side lengths $\{30, 40, 50, 60, 70, 80, 90\}r_c$ and density $\rho = 3.0$. The simulations were performed on a CUDA Tesla C1060 card, which has 240 streaming processor cores running at 1.3Ghz with 4Gb of onboard RAM, connected to a desktop machine with an Intel Core 2 Duo 6320 processor and 2Gb of RAM.

Figure 4.13 shows that the execution time for each method call on the CPU is independent

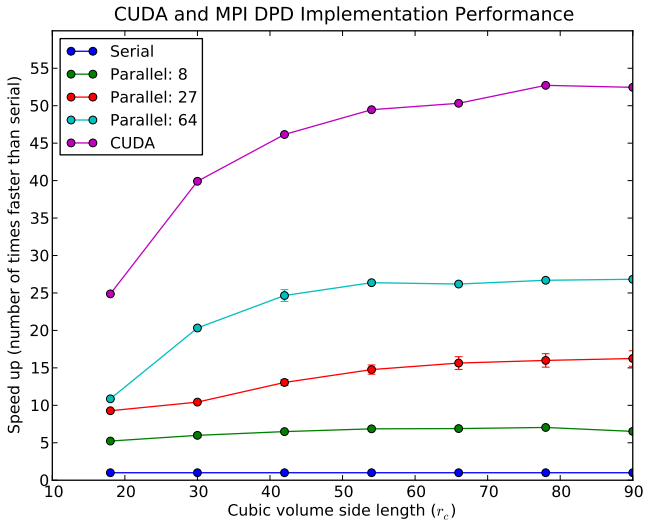


Figure 4.12: Performance of the CUDA and MPI versions of DPD: The speedup relative to the serial version of the code for increasing simulation volumes is compared for the CUDA, single processor and MPI based parallel using 8, 27 and 64 nodes. Three runs were performed for each side length and the data points indicate the mean speedup (error bars indicate the standard deviation, if large enough to be visible.)

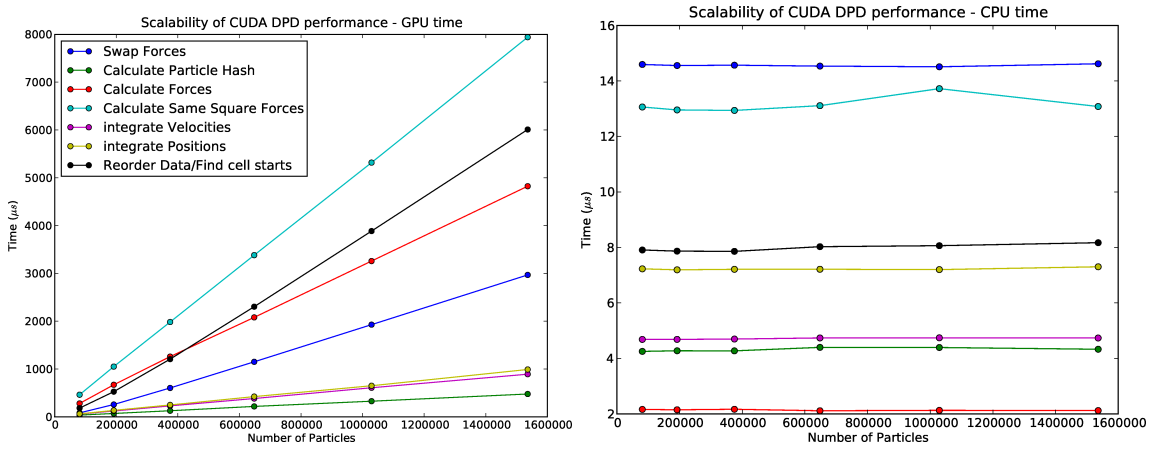


Figure 4.13: Execution time of each kernel on the GPU (top) and CPU (bottom), the legend in the GPU time plot also applies to the CPU plot.

of the number of particles, and does not exceed 15 microseconds. The execution time of the methods on the GPU scales linearly with the size of the input. Small kernels with few loops/registers, such as the integration methods and the particle hash calculation execute in less that a millisecond even for systems with over one and a half million particles. Note that the "Calculate Same Square Forces" kernel, which performs the force calculation between particles within the same cube takes longer to execute than the "Calculate Forces" kernel, which performs the force calculation between two

adjacent cubes. The explanation for this is that the latter method calculates forces for half of the particles in the simulation (the forces for the other particles are calculated in another call to the same method with the shift parameter altered). Although the calculate forces method does not require the most execution time on the GPU, it must be called 13 times for all particle forces to be calculated within a timestep, and so makes the largest contribution to the overall runtime.

The amount of time spent in each method per timestep, estimated from the results shown in Figure 4.13 is shown in table 4.1.

Method	Calls Per Timestep	CPU/GPU time μs
Swap Forces	1	14.53/1151.64
Calculate Particle Hash	1	4.397/218.47
Calculate Forces	13	54.86/54098.98
Calculate Forces (Same Cell)	1	13.7224/3382.65
Integrate Velocities	3	14.19/690.42
Integrate Positions	1	7.21/423.83
Reorder Data	1	8.03/2304.52
Total		116.94/62270.51

Table 4.1: Total execution time spent in each method for calculation of a single timestep in simulation of 60^3 cubic volume containing 648,000 particles

The table shows that almost 90% of the execution time is spent in the force calculation methods. Another measure of how efficiently the graphics hardware is utilised by a particular kernel is the thread occupancy, which is the ratio between active warps to the maximum number of warps supported by a multiprocessor. Analysis of the thread occupancy for each kernel showed that all kernels had 100% occupancy, except for the calculate adjacent forces kernel, which had a thread occupancy of 50%. The reason for the reduced occupancy is that the kernel requires 30 registers, which limits the possible number of active threads per multi-processor. By reducing the number of registers used in the thread to improve occupancy, performance may improve. However, initial attempts to move particle data into shared memory resulted in a drop in overall performance.

4.4 Modelling Chemical Interactions

DPD is often used to study the dynamics of soft matter systems such as membrane structures, colloids and suspensions. Such systems are generally in chemical equilibrium and the standard DPD formalism does not include a mechanism for chemical reactions. Several extensions have been made to Dissipative Particle Dynamics to permit chemical reactions to occur within the simulation. For

example in [86] Fellermann et al. added a form of chemical reaction where two particles collide and the types of the particles are changed, but in this scheme reaction probabilities are proportional to inter-particle distance. Dynamic bonding DPD (DDPD) was introduced in [42] which involves the formation of model covalent bonds based on Hookean spring potentials, permitting the formation of polymers. These chemical reaction schemes enable the specification of two different types of reaction. However, there has been little attention given in the literature to how the rates for these reactions might be specified with relation to experimental observations or the relationship between collision reactions in DPD and other reaction simulation techniques. The previously proposed reactions schemes added to DPD fall into two categories:

- Polymerisation reactions, in which reactions occur when beads come into close proximity and a bonding force is added to those particles with a certain probability. The formation of the bond may or may not be reversible.
- Type change reactions, in which the reactions occur when beads come into close proximity and the types of the two reactants are changed with a certain probability to the types of the reaction produces.

Polymerisation reactions have so far only been used for simulation of simple dimerisation reactions, and formation of more complex polymers would be difficult to implement as it would be necessary to limit the reactions so that they may only result in the formation of some predefined polymer template. It is also not clear how reactions between polymers should be modelled. The benefit of this approach is that the resulting polymers are not limited in size and shape, and if they are reactive with other polymers then reduction in collision probability due to the polymer's larger mass will be captured explicitly.

Type changes reactions have been applied in complex artificial life simulations, and are more easily implemented as a reaction involves simply changing the types of the particles. As the total number of particles in a DPD simulation should remain constant, this type of reaction scheme only permits first and second order reactions which are balanced, meaning that for a first order reaction, one product bead is produced from one reactant bead, and for a second order reaction two product beads are produced from two reactant particles. The constraint on the number of particles can be relaxed, allowing reactions in which a single reactant bead produces two products, or where two reactant particles produce a single product particle as long as the effect of adding/removing particles will be negligible. The difficulties of adding chemical reactions to DPD are threefold:

- Energy is not conserved by the method, excess energy released by reactions will be dissipated within a small amount of time.
- The DPD beads typically represent several molecules of a substance rather than individual atoms. It is not clear what a reaction occurring between two beads represents.

- Since reaction schemes are often arbitrary, it is difficult to determine how to convert rate constants from other models or observed data.

The reaction scheme for DPD which is proposed in this thesis does not address the first problem of energy conservation, and so DPD models of reactions cannot represent the endothermic and exothermic effects of reactions occurring in the simulation. The scheme addresses the second difficulty by simply assuming that when reactions occur between beads, the physical interpretation of the event is that the number of reactant molecules represented by the DPD particles react all at once, to produce a bead or beads which represents the output of those reactions. The key benefit of the reaction scheme proposed in this work is that it addressed the final difficulty in this work, as it permits straightforward conversion between reaction rates from other modelling tools (e.g. ODEs, SSA) and DPD.

Unlike in other schemes the probability of reaction in the proposed scheme is not determined by the distance of the two beads from one another, and instead a reaction may occur whenever two compatible beads come within a given radius of one another with a certain probability that is specified by the reaction rate parameter. By modelling reactions in this way, it is possible to apply theoretical results from collision theory to calibrate the reaction rates in the DPD simulation with those from physical observations. The results indicate that the reactant collisions are governed by gas phase kinetics, with a reduced mean particle velocity. Furthermore, a reaction scheme which involves changing bead types in this way encounters another problem when representing unbalanced reactions (e.g. those which have an unequal number of reactants and products), as this means that particles must be removed or added to the simulation volume, which means that the DPD simulation is no longer performed in an NVT ensemble. A simple method to overcome this problem involves adding "implicit" solvent particles to balance the reaction.

The proposed reaction scheme supports the following types of first and second order reaction. Unbalanced reactions are realised in the simulation by implicitly balancing the reaction with an additional solvent reactant or product. Reactions such as these might represent a complexation and subsequent decomplexation, or the forming and breaking of a ionic or covalent bond. In the case of the 1 input-2 output reaction, the neighbours of the input particles are searched until a solvent particle is found, which becomes the second output particle. Despite the introduction of the solvent particle input, the 1 input-2 output reaction is still first order, and so the reaction is only attempted once per timestep rather than once per collision between the input type and a solvent particle. For the 2 input-1 output reaction type, the type of one of the input particles is changed to solvent, and the other to the reaction product type.

Second order reactions in the reaction scheme occur as follows: when a pair of reactable particles "collide" a random number is generated and if this number is less than the reaction rate parameter c_{dpd}, then the reaction occurs. As the potentials in DPD are soft, the collisions are not elastic, and it is even possible (although highly unlikely) that two beads may occupy the same

Inputs	Outputs	Description
1	1	1st order state change
2	1	complexation reaction - two inputs form a single molecule, complex etc.
1	2	decomplexation reaction - a single reactant decomplexes into two components
2	2	2nd order state change

Table 4.2: Table of the possible reactions in the CUDA DPD type reactions implementation

position. Therefore a collision in the proposed reaction scheme simply means that the two reactants move within each others "reaction radius" which is specified for each reaction as a parameter r_{react}. By choosing a reaction radius within which reactions occur with a certain constant probability rather than relating reaction probability to inter-particle distance, it is proposed that the kinetics of reactions in DPD can be characterised by simple gas phase kinetic theory, with a reduced average particle velocity. Figure 4.14 shows a collision between two particles, the reaction radius and the force interaction radius are shown.

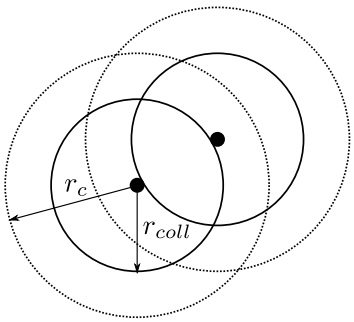


Figure 4.14: Collision between two particles: When two particles (whose centres of mass are represent by a black dot) are within the collision radius r_{coll} of one another, then a collision is considered to have occurred. The length of the collision radius is a parameter which is configurable for each reaction, and must be less than the radius of force interaction r_c.

4.4.1 Reaction Rates For Second Order Reactions

The rate at which a reaction occurs depends on the reaction parameter c_{dpd}, the concentration (or rather the number of reactant pairs in the fixed volume system), and the rate the reactants diffuse within the solvent (which is a function of the 3 force parameters). The conservative parameter is a factor in the rate of reaction as if the reactants have a large conservative potential with particles of the solvent type, then the reactants will be forced into close proximity by action of the model hydrophobic effect, and this decreases the volume in which the reactants move, increasing the probability of

collision and the rate of reaction.

In rate equation models of chemical reactions, the reaction rate $\mathcal{R}$ of a reaction occuring between two distinct reactants is defined in terms of the concentrations of the two reactants, and the rate constant (typically denoted k), which is dependent on properties such as the temperature and the activation energy of the reaction.

$$\mathcal{R} = k[Y][Z] \tag{4.2}$$

Equation 4.2 shows the reaction rate for a second order reaction. where k is the rate constant, and $[Y]$,$[Z]$ are the concentrations of chemical Y and chemical Z respectively. If specified in terms of the number of particles rather than the concentration, the reaction rate becomes

$$R = cYZ \tag{4.3}$$

where R denotes the number of products from the reaction and c is the reactions per particle pair per unit time and Y,Z are the number of particles of type Y and Z respectively. The c constant can be understood as

$$c = n_c p_r \tag{4.4}$$

where n_c is the number of collisions per particle pair per unit time, and p_r is the probability that a collision will result in a reaction. Therefore the rate constant for second order DPD reactions is the probability of a collision between a particle pair resulting in a reaction, p_r.

$$c_{dpd} \equiv p_r \tag{4.5}$$

After determining the rate of particle collisions in the DPD simulation, it is possible to convert rate constants for 2nd order reactions derived from observation or other modelling techniques into DPD reaction rate parameters, if the length and timescales of the simulation have been given a physical meaning, and assuming that the collisions in the artificial chemistry obey the ideal gas laws. The volume of a DPD particle in the simulations is $90\mathring{A}^3$, equivalent to that of three water molecules. Groot and Rabone [111] showed that using this mapping the unit distance r_c is $\sim 6.4633\mathring{A}$ and the time unit τ can be interpreted as being $\sim$ 88ps. An a parameter of $a = 78$ reproduces the compressibility of water at room temperature (20°C). Knowing the physical length, time and energy scales of the system allows a comparison to be made with the theoretical probabilities calculated with the Maxwell-Boltzmann equation, by using the relation between the collision volume of particle pairs and the average velocity of the particles, from [104]:

$$\overline{\delta V_{coll}/V} = V^{-1}\pi r_{12}^2 \overline{v_{12}}\delta t \tag{4.6}$$

Where V_{coll} is the collision volume swept out by a particle in the infinitesimal time increment δt, V is the system volume, r_{12} is the distance between the particles and v_{12} is the relative velocity. Gillespie states in the same reference:

> For Maxwellian velocity distributions the average relative speed $\overline{v_{12}}$ will be equal to $(8kT/\pi m_{12})^{1/2}$, where k is the Boltzmann's constant, T the absolute temperature, and m_{12} is the reduced mass $m_1 m_2/(m_1 + m_2)$.

Collisions in DPD are inelastic due to the soft repulsive nature of the conservative force, but as a collision is considered to have occurred whenever two particles come within the force interaction radius, the reaction dynamics should be similar to those of systems with elastic collisions and observations of DPD showed that this was the case.

To investigate the dynamics of the reactions, simulations of a 2nd order reaction with a rate constant $c_{dpd} = 0.01$ were performed in cubes of increasing volume. Reactions were recorded at each timestep and the particle pair collision probability per time unit τ was then calculated at each timestep with the following formula.

$$P_{coll} = \sum_{i=0}^{N_s} \frac{R_i}{X_i Y_i} dt \tag{4.7}$$

Where i is the timestep in relation to the current time unit, N_s is the number of steps per time unit, R_i is the number of reactions occurring at timestep i, X_i and Y_i are the amount of X and Y reactants at the start of timestep i.

The relationship between the probability of collision and the volume simulation was investigated through experimentation by simulating a 2nd order reaction in increasing volumes.

$$X + Y \rightarrow Z \tag{4.8}$$

The concentration of X and Y reactants in each simulation was set such that 10% of the particles were of type X and 10% of the particles were of type Y and simulations were performed of cubic volumes with side lengths: $\{10,20,30,40,50,60,70,80\}r_c$ and each simulation was performed for 1000 timesteps after allowing the system to equilibriate for 1000 steps in which no reactions were allowed to take place. 100 runs were performed at each volume, and the probability of particle pair collision was calculated from the mean time-series at each timestep by recording the number of reactants at the start of each timestep and the number of reactions that occurred during each timestep which was used to estimate the reaction rate with the following formula

$$r = \frac{\sum_{i=0}^{i=N} \sum_{j=0}^{j=n} r_j X_j Y_i c_{dpd}}{N} \tag{4.9}$$

Where N is the length of the simulation in DPD time units, n is the number of timesteps in a DPD time unit, X and Y give the number of X and Y particles at the start of each step, r_j is the number

of reactions that occurred in step j and c_{dpd} is the DPD reaction rate.

Figure 4.15 shows the observed relationship between the probability of collision and the system volume from simulation and the calculated estimates, which is compared with the theoretical collision rate calculated from the formula 4.6.

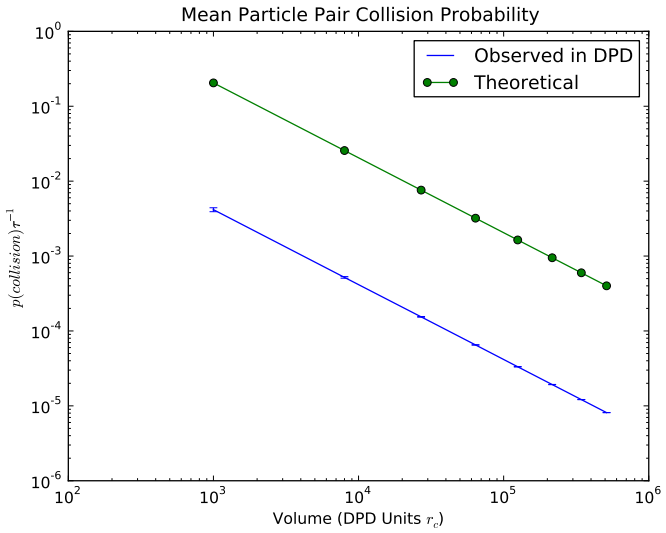


Figure 4.15: The estimated particle pair collision rate, for a range of different cubic volumes in DPD

The results show that the DPD collision probabilities decreases exponentially with exponentially increasing volume, and the figure indicates that the relationship between the volume and the collision probability obeys formula 4.6 but with a reduced mean relative speed. Regression analysis on the DPD results showed that the mean relative speed for DPD colliding particles was 1.324 (in reduced units), compared to the calculated value of 2.257.

Figure 4.16 shows a comparison of the time series of a first order reaction ($X + Y \rightarrow Z$) in DPD and in Gillespie's SSA (in which the reaction rate is parameterised from Equation 4.6) for cubic volumes with side lengths 20-80. There is a good correspondence between the time series in DPD and SSA indicating that the observed collision probabilities are accurate. The DPD reaction rate parameter c_{dpd} can then be set accordingly to model the dynamics of a collision based reaction. The $p(collision)$ value also establishes the upper bound for the reactions which can be modelled in the given volume (e.g. no second order reaction can occur more quickly than the particle collision rate).

The three different versions of the DPD simulator (serial, parallel and CUDA) are the core of the DPD toolkit, which contains programs for the display, analysis and modification of data files. These tools are described in more detail in the next section.

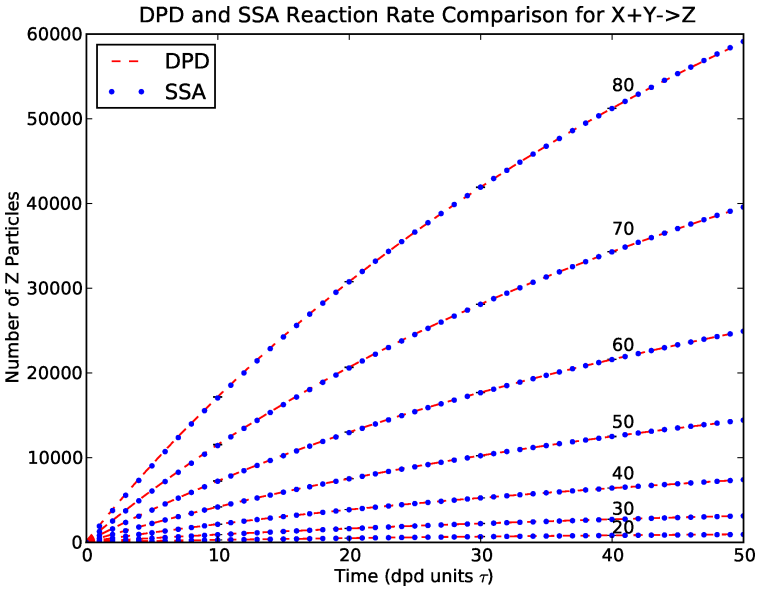


Figure 4.16: Chemical time series for the reaction $X + Y \rightarrow Z$, each line shows the number of Z particles over time for increasing simulation volumes, the dashed line shows the time series for the SSA simulation of the same volume using the rate determined from the formula, the time series is the average of 10 runs

4.5 Display and Analysis

It is often useful to enable the modeller to examine, view and rotate the self-assembled structures in three dimensions, to gain an intuitive understanding of the process of formation and the resulting equilibrium state and to gauge qualitatively the overall behaviour of the system. With this in mind, a display program was added to the DPD toolkit, which enables the modeller to view and investigate the output from the DPD simulation programs. The DPD display program is written in C++, and renders a display of the particle data by interfacing with the *OpenGL 3D* graphics libraries. The display tool enables the modeller to view the particles and polymers in each recorded time step, and display the results of analysis of the data (e.g. view detected objects) and temporarily translate and rotate the data to gain an intuitive understanding of what is happening in the simulation. Figure 4.17 shows a screen shot of the DPD display tool, which is displaying timestep from a vesicle self-assembly simulation.

The display tool also enables the extraction of objects from the simulated data, by first identifying objects (i.e. vesicles, micelles etc.) using morphological analysis techniques, and then saving the resulting objects to data files. The objects are stored in the same format as the simulation data, and so they can be viewed with the dpd display tool in the usual manner, and stored for later recombination into a new initial state using the initial state creation tools. The user interface code the display tool is constructed around the *libGLUI* graphical user interface library, which is an open source library enabling the programmer to create graphical user interfaces for 3D systems by placing

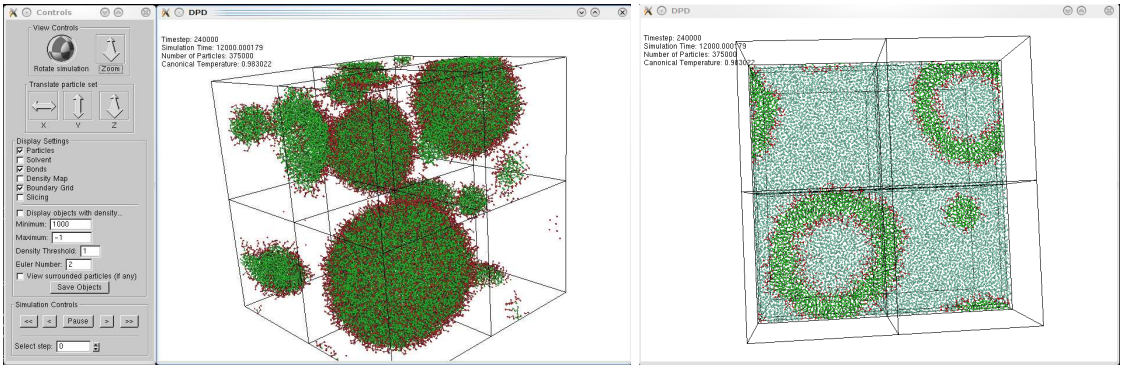


Figure 4.17: The DPD toolkit display tool. The left image shows the display of a timestep containing several self-assembled vesicles, with the control panel enabling manipulation of the data for easy viewing. The right image shows a sliced view of the same timestep, with the blue solvent particles displayed.

a number of predefined controls on the user interface, such as buttons, check boxes etc. The display tool also reads configuration from a configuration file using the *Libconfig* library, which enables the user to specify the colours used to display the different particle types and the behaviour of the object analysis code.

The identification and extraction of the membrane objects was automated so that objects of interest (such as vesicles) can be discovered based on the positions and distributions of polymers and particle types. These objects can then be saved as object data files if required. The development of algorithms for the analysis and characterisation of self-assembled objects based on the locations and chemical species of the particles, in a manner which is efficient enough so that each timestep in a large data file with unknown contents can be scanned presents a number of challenges and for this work a local search based heuristic method, akin to a three dimensional flood fill algorithm, was employed to identify the membrane objects within the simulation.

During the initialisation of the object detection algorithm, the particles in the simulation space are binned using cell tables as described in section 4.3.1, which results in a spatially indexed data structure listing the particles contained within each cubic sub-volume. A second spatially indexed data structure is constructed, which maintains a Boolean value for each cell in the cell table, indicating whether the cell has been visited by the algorithm or not. The search begins by checking the "visited" flag for each cube in the cell tables sequentially. If the visited flag is false, then the number of particles which are of the hydrophobic particle type within the cell are counted, and the visited flag is set to true. If this particle count exceeds a given threshold (which is a configurable parameter of the algorithm) then a patch of membrane is assumed to be located within the cell, and an object has been found. Once this occurs, then the polymers which own the particles within the membrane cell are added to a set, and then the algorithm searches the cells neighbouring the cell of interest. For each of these cells, if the cell has not been visited and contains membrane particles,

then the polymers owning those particles are added to a set, and the neighbouring particles of that cell are also searched. This process continues until no more neighbouring cells can be found that contain membrane particles, at which point all of the polymers of the object have been found and the polymer list is stored in the list of discovered objects. The algorithm continues searching the unvisited cells sequentially, until all cells in the table have been marked as visited. The flood fill algorithm returns a list of objects found in the simulation, where each object is composed of a set of polymers. Post-processing can be performed on the resulting objects to remove from the list those objects which are composed of too many or too few polymers, which allows the modeller to ignore those objects which are not of interest. Low and high thresholds for object size can be specified as parameters to the algorithm.

For automated analysis of the data, the next requirement is to determine, based on the positions of the polymers which make up the object, what type of self-assembled structure the object represents. A metric which uniquely characterises the shape and size of the object, such that the type of object (e.g. micelle, bilayer, vesicle etc.) can be determined is therefore required so that vesicles within large volumes of outputted data can be easily identified and extracted. More specifically in the case of vesicles it is also useful to determine what particles or polymers are contained within the vesicle core, so that they can be extracted and stored along with the vesicle membrane polymers.

The instantaneous shape of an object may vary quite considerably due to the flexibility of the membrane. Depending on the pressure, membrane composition, thermal fluctuations and stresses on the vesicle, it may assume a large number of different shapes at mechanical equilibrium. However, regardless of the varying shape, some properties of a structure remain constant. For example, in order to be a vesicle, the topology of the object should be that of a volume of fluid which is completely encapsulated within a membrane. Insights into the characterisation problem can be gained from the field of morphological image analysis. The characterisation of the morphology of an image or section of an image is a well researched problem in image analysis and finds application in a number of different areas of research, such as the automated analysis of the output from magnetic resonance imaging machines [193].

The Minkowski functionals are a set of metrics which are used to topological and geometrical properties of a structure, and have been applied in analysis of data in a number of different contexts [185], including the identification of vesicles in molecular dynamics simulations[176]. The essence of the Minkowski functionals analysis is that it is only necessary to consider three metrics (four in three dimensions) to fully characterise the morphology of an object. These measures are ideal for characterising the objects contained within a previously unexamined data file generated from a DPD simulation, as it is not necessary to make any prior assumptions about the kind of objects which are contained within the simulated volume when calculating the metrics. This enables an algorithmic analysis of the simulation space, which can produce machine or human readable data indicating the objects which have formed.

The functionals can be calculated in scalar, vector or tensor forms and it is the three

dimensional scalar metrics that are used to characterise the DPD objects. The algorithm is described in [184], and operates by considering each voxel in the object to be a union of the cubes interior, open faces, open edges and open vertices. If the n_0 is the number of open vertices, n_1 is the number of open edges, n_2 the number of open faces, and n_3 the number of open voxels, then the functionals can be calculated as follows:

$$V = n_3 \tag{4.10}$$

$$S = -6n_3 + 2n_2 \tag{4.11}$$

$$2B = 3n_3 - 2n_2 + n_1 \tag{4.12}$$

$$\chi = -n_3 + n_2 - n_1 + n_0 \tag{4.13}$$

where V is the volume, S is the surface area, B is the mean breadth, and χ is the Euler characteristic.

The first stage of the Minkowski analysis algorithm, requires that the object volume is somehow discretised to produce a binary image representation of the object membrane structure. In three dimensions, this can be achieved by "voxelizing" the membrane, producing a three dimensional structure of volume elements, which are coloured black if the volume element contains membrane particles, and white otherwise. During the flood fill algorithm, a voxelisation of this sort is performed when the "visited" data structure is created, and so the voxels relevant to each object are stored in the object data structure. These voxels can then be used to perform the analysis.

Figure 4.18 shows a screen capture of the results of this process, with different coloured voxels showing the binary 3D image representations of a number of different objects.

One problem with applying the Minkowski functionals to the object characterisation problem is that the technique is very sensitive to cavities within the membrane structure. These cavities can appear due to the voxelisation of the object particles as it may be the case that the hydrophobic membrane particles are not quite evenly distributed, due to short time scale membrane fluctuations, or that the resolution of the discretisation is too high, such that some voxels which are located within the membrane volume will not be marked as membrane voxels. If this occurs in the centre of the membrane, then a small cavity will be present within the 3D image of the membrane volume. These cavities will result in the incorrect characterisation of the object's topology, and so it is necessary to apply a post-processing step after generating the image from the object. Figure 4.19 shows the cavity problem in a two dimensional image, and the result of applying the dilation algorithm to that image. The cells which are within the bounding box of the object are scanned, and if a cell is marked as not being a membrane cell, but is surrounded by membrane cells on more than n sides (where n is a configurable integer parameter between 1 and 6), then it is marked as being a membrane cell. The scan is performed repeatedly, until no changes are made to the image of the object. The downside of this heuristic approach is that it alters the values for the other functionals (e.g. surface area and volume). However the most important value when characterising the object is the value indicating

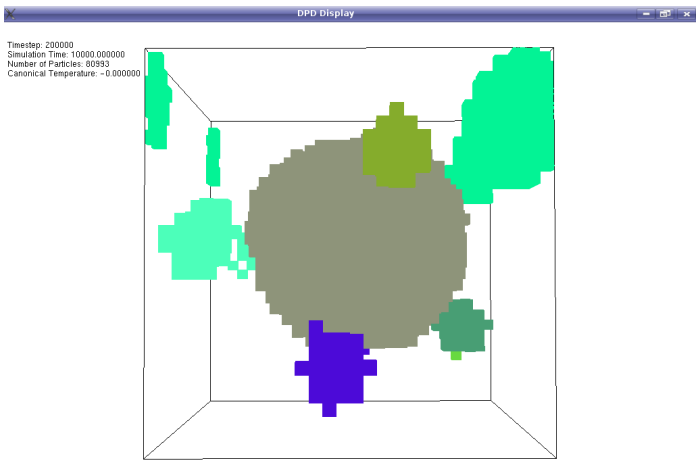


Figure 4.18: A screen capture from the display tool, showing the identification of objects in the simulation. The particles are binned using the cell tables algorithm and objects are identified using the algorithm described above. For each object, the cells which contained a majority of hydrophobic particles were determined to be membrane voxels, the 3D voxel was displayed as a coloured cube, with each voxel for an object being assigned the same colour, a spherical vesicle at the centre of the volume is coloured grey, and other objects surrounding the vesicle are small micelles.

the object topology, which is the objects Euler number, described above.

4.5.1 DPD Initial Simulation State Creation

To simulate vesicle computing systems described as stochastic P systems, it is necessary to create the initial state of the system. This involves the placement of previously formed vesicle objects, and the modification of the vesicle inner volumes to create a set of vesicles, surrounded by solvent particles and encapsulating the necessary vesicle computation functionality. The *Initial simulation state creation* tool was developed for the purpose of creating these initial states, which are then used as the initial particle configuration for the vesicle computing simulation.

Often it is desirable to experiment with a simulated vesicle which has certain properties, such as encapsulating a certain volume of fluid, or containing a certain number of amphiphiles in the membrane. As the process of vesicle self-assembly is not deterministic, attempting to create vesicles with the desired properties from random configurations would require waiting for such a configuration to occur by chance. Instead, it is possible to assemble membrane structures artificially, in an algorithmic fashion, and then to manipulate and utilise these artificial structures in the same way as self-assembled structures. The functionality for the artificial formation of vesicles is included in the *dpdtimestep* tool which enables the placement of amphiphiles in vesicle configurations, so that vesicles of any size and volume can be generated. Figure 4.20 shows an artificial vesicle created with the dpdtimestep tool.

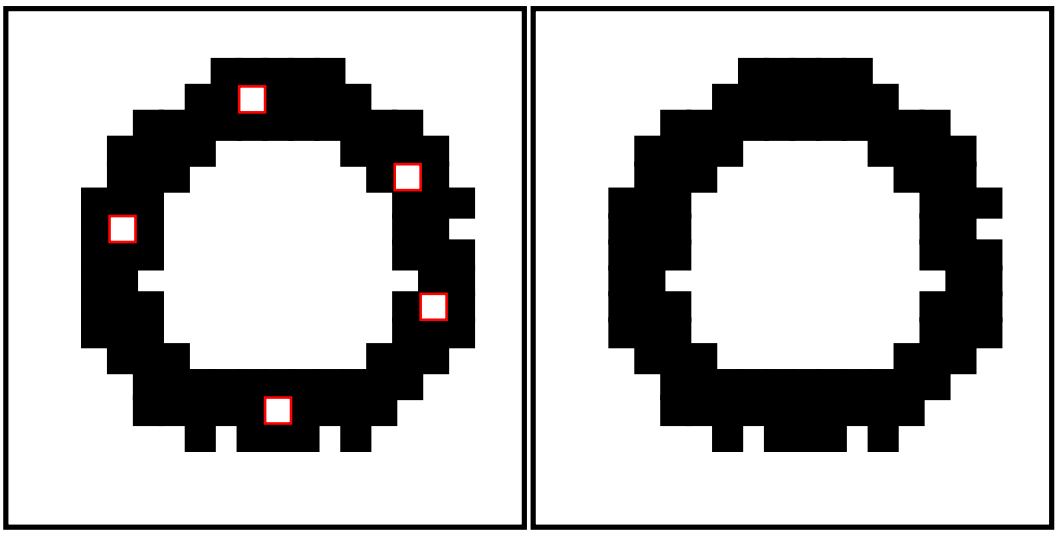


Figure 4.19: An illustration of the cavity problem in two dimensions, the left image shows a circular membrane structure with several cavities within the membrane (the pixels which have a red coloured border). If these cavities are present during the Minkowski functional analysis, then the Euler number will be incorrect for a vesicle and the vesicle will not be detected. The right figure shows the same image after the dilation algorithm has been applied.

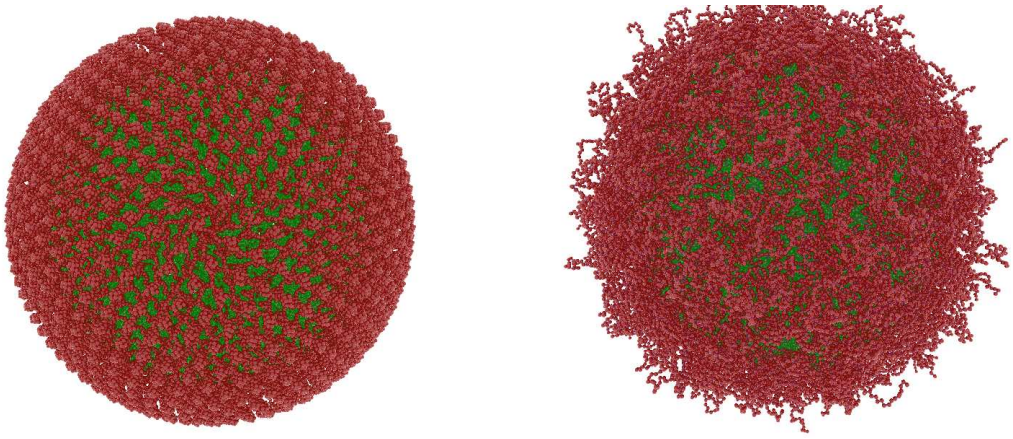


Figure 4.20: A vesicle configuration created with the dpdtimestep tool by the algorithm described above, the left image shows the spherical configuration of the amphiphiles that make up the membrane, and the right image shows the result of simulating the same vesicle membrane structure for 200000 time steps. The amphiphiles have relaxed from the initial spherical configuration into a stable vesicle.

4.6 Configuration and Data Storage

The input to the DPD simulation tools is a config file (an example configuration file is given in appendix A). The format of the configuration file is the same for the three different versions of the program (with an additional processor configuration section added to the parallel configuration, which is ignored by the other versions), meaning that configurations generated for one version can be used for either of the other two versions with little or no modification.

The loading and parsing of the configuration files is handled by the external *Libconfig* library[2], which is an open source configuration library, released under the GLPL license. The library supports configuration files in a custom format which is described by a BNF grammar, and provides a concise specification of configurations in a text based format supporting hierarchical configuration with full support for lists and tuples, and automatic type inference and checking for the configuration files.

The output from the DPD simulation programs are data files which contain the following information recorded from the simulation (full details are provided in appendix A).

- Information regarding the DPD simulation tool used to perform the simulation (e.g. the platform, such as CUDA, and the precision of the floating point numbers)
- The entire configuration file used to perform the simulation.
- The configuration of the polymers in the simulation (if appropriate) depending on whether dynamic bonding is enabled. This information may be recorded once at the beginning of the data file, or once every time step.
- For each recorded timestep (note that the DPD configuration system allows the user to specify when to start and stop the recording of particle data, and the interval of collection), the data for each particle in the system (positions, velocities and types).

The data file classes provide an interface to the raw binary data (which will differ in terms of the precision of the floating point numbers, depending on which platform is used to perform the DPD simulation), which means that regardless of the format of the data, it can be extracted in a coherent form for viewing and analysis by other tools in the DPD toolkit. The choice of programming language when implementing the DPD framework was somewhat constrained by the performance requirements, and the intended environments on which the code will be executed. As the intention when designing the software was to perform runs on the University of Nottingham cluster and on GPUs, the application must be compatible with the available languages and libraries provided as part of the cluster operating system as well as being cross platform and enabling the programmer to "get close to the hardware" when low level optimisations were required. The two languages which seemed most appropriate for this purpose were C and C++. Of these two, C++ was chosen as it contains powerful object oriented design features, while still enabling low level optimisations and

supported a wide range of libraries for easy management of peripheral aspects of the DPD software, such as configuration file and basic sorting algorithms etc. C++ is also fully supported by all of the GPU programming environments available.

For certain parts of the simulation toolkit, such as the plotting of the DPD data, the performance of the code was less of a requirement than the development time of the code. In these instances, the Python programming language was used. Python is an interpreted scripting language with a clear and concise syntax, with a wide variety of scientific plotting libraries available which enables rapid scripting of plots from the output statistics from a DPD simulation.

4.7 Summary

In this chapter, the DPD toolkit was described in detail. Several different applications were created which enables the modeller to specify, simulate and analyse liposomes at the mesoscale. The DPD toolkit forms part of a larger simulation and modelling framework described in the previous chapter, enabling the exploration of the mesoscale aspects of vesicle computation, such as the membrane composition and dynamics.

The simulator was optimised to enable large volume simulations of vesicle formation and dynamics to be performed and two different parallel implementations of the DPD software were described, the MPI distributed memory cluster implementation, and the CUDA implementation. The performance of these implementations was compared with the serial version and the CUDA version outperformed the serial version more than fifty times and was faster than the MPI version running on 64 nodes in the cluster. Since a single graphics card is several orders of magnitude cheaper than a 64 node cluster, it is clear that CUDA offers a very good price to performance ratio, and enabled the routine simulation of systems in DPD which were similar in volume to some of the largest reported in the literature. The CUDA implementation does have some drawbacks however, such as the lower precision floating point numbers. Comparisons between observable values in simulations performed with double precision floating point with the serial version, and single precision floating point using CUDA, indicated that the loss of precision in CUDA does not result in a significant enough alteration in the observed values to alter the conclusions which are presented in this thesis.

Chapter 5

Vesicle formation in DPD

Program designs in vesicle computing should be able to utilise combinations of vesicles, either of the same composition or of different composition depending on the requirements, to perform a given computation. Since the aim of this work is to investigate the feasibility and behaviour of vesicle computations, the ability to express vesicle computations with a variety of vesicle compositions and topologies is a clear requirement. In this chapter, the parameters and analysis of a vesicle library, containing a large number of self-assembled membrane structures, are provided, as well as results from vesicle formation experiments.

5.1 Introduction

The MIT repository for biological parts is a website which enables researchers to deposit and access many different genetic building blocks which can be recombined to enabled rapid development of synthetic biology systems, both in vivo and in silico. In this chapter, I describe the development of a similar parts library for vesicle computing simulations, which enables the rapid prototyping of computations by combining pre-assembled membrane structures. One of the goals of this research is to create a library of liposome computation membrane elements, similar to the biobrick parts in the MIT biobricks repository of engineered genes and gene regulatory networks. The vesicle library contains a large number of membrane objects extracted from the simulation data, which in combination with the DPD simulation software, enables a modeller to easily create new simulation initial states by selecting objects from the library and placing them in a new volume, possibly altering the composition of the vesicle core volume to investigate a specified scenario.

By building and maintaining a library of vesicles in this way, simulation times can be reduced, as it is no longer necessary to rely entirely on the process of self-assembly to generate a desired membrane computing structure. Vesicle objects which may be stable at equilibrium but may not form as a result of the self-assembly process can also be created artificially and stored for later use. The vesicle library also enables the rapid combination of objects which have formed from a variety of different parameter sets and model amphiphiles, and there are a number of model

amphiphile parameters which result in vesicle formation that have been reported in the literature for the DPD method.

Many of the vesicle simulations described in this thesis involve the self-assembly of membrane objects from random initial configurations of amphiphiles in solvent. Automated detection and analysis of self-assembled structures in simulation data allows the extraction of those structures so that they can be added to the vesicle library. Integrated vesicle identification and analysis allows the automated processing of the large volumes of data generated from the parallel implementations of DPD, allowing the collection of a large number of vesicle objects of different volume and composition which can be reused to create new vesicle computing simulations and components.

The addition of this functionality poses a number of technical challenges, for example identifying vesicles from particle and polymer position data. The automated analysis is included in the DPD toolkit described in chapter 4, which contains a flood-fill based membrane identification algorithm, which is combined with an analysis method based on the Minkowski functionals. It is also desirable to be able to generate artificially membrane structures such as vesicles and bilayers to simulate structures which may not form by self-assembly and this functionality is also included in the toolkit.

Finally, the storage of the vesicles and membrane structures permits the analysis of the statistical distributions of vesicle properties such as the encapsulated volume, surface tension and membrane thickness. Such an analysis enables comparison of the results of the vesicle formation with lab data.

In the rest of this chapter the different parameterisations for vesicle formation in DPD are reviewed, and the vesicle computing object library is described in more detail. The results of a large scale study of DMPC vesicle formation are presented, as well as the formation of stable PEO-PEE vesicles from artificial configurations, and the derivation of a new DPD polyethylene oxide-polylactide (PEO-PLA) diblock copolymer model.

5.2 Simulation of Bilayers and Vesicles in DPD

One aspect of soft matter modelling in which DPD has been shown to be very useful is the simulation of amphiphilic molecules, which self assemble due to hydrophobic and polar interactions with solvent into a number of different phases depending on the temperature and amphiphile concentration. These phases include micelles, membranes and vesicles. Lipid based bilayer membranes are common in nature and form the basis of the cell membranes that delimit the intracellular components of cell from the extracellular environment. The formation of membranes and the dynamics of membrane processes such as fission and fusion [55] occur at length scales that are often shorter than the wavelength of light, and at timescales that are of the order of a few microseconds. This makes direct observation of these processes difficult, and so simulation techniques have been employed to gain a better understanding of membrane formation.

5.2.1 Simulation of Bilayers

Early simulations of bilayers were performed in [106] and [235] using molecular dynamics and a Monte Carlo method with a 2-dimension lattice model respectively (see [90] for a review of early MD membrane simulations). One of the first simulations of a bilayer in DPD was performed by Groot et al. [109] who simulated lamellar sheets of di-block copolymer membrane. Venturoli et al. [258] performed simulations of a small patch of bilayer, composed of 100 phospholipids, using a Monte Carlo scheme to ensure that the tensionless property of natural bilayers was reflected. The pressure profile of the membrane was then calculated and compared to results from a lattice model [47]. The results from the DPD simulations were in good agreement with the lattice model results. The Monte Carlo scheme for ensuring the correct membrane tension outlined in this paper has been employed in a number of later works [146]. The DPD model has since been used in a number of studies of the elastic properties of membranes such as tension and bending modulus, producing values for these properties that are close to the experimentally observed values. One attempt to use the DPD model to gain a better understanding of biological membrane processes is given by Groot et al. [111] in which a simulation of the effects of nonionic surfactants on bacterial cell membranes was performed. The premise of this work was that biocidal compounds, although wide ranging in structure and composition damage membranes in similar ways, indicating that the process involved may not involve complex interactions between individual molecules at the atomic scale, but may be more of a physical process involving the interaction between the membrane and amphiphiles with different structures. The authors performed a simulation of a bilayer composed of phosphatidylethanolamine amphiphiles, in the presence of mole-fractions of non ionic surfactants between 10%-100%. It was found that the presence of surfactant reduces the resistance of the bilayer to tension, and led to stable pore formation at 90%-100% mole fraction. Simulations of biological membranes at a larger scale were performed in [231] and of unilamellar sheets formed from di-block copolymer melts in [109, 110].

Bilayer membranes have also been shown to exhibit a shift to an interdigitated phase, in which the amphiphile tails of one mono-layer can infiltrate the voids between amphiphile tails of the other mono-layer. The effect of head group repulsion, temperature and amphiphile structure on the interdigitation of dipalmitoylphosphatidylcholine (DPPC) bilayers was investigated in [146] by using DPD simulations of patches of bilayer. It was found that for single tail amphiphiles, increasing the repulsion between the head groups caused interdigitation, whilst for double tail amphiphiles, it was necessary to increase the size of the headgroup by adding another head bead to cause interdigitation to occur. Further work by Kranenburg and Smit [145] was then carried out to investigate the affect of alcohol on lipid bilayer membranes, in which it was shown that the alcohol molecules caused interdigitation by preventing the hydrophobic tails of the interdigitated bilayer from being exposed to the solvent. At lower concentrations of alcohol a rippled phase was seen to form, similar to the ripple bilayer phase that was investigated by the same authors using DPD [143].

Fusion between lipid membranes is a topic of interest in biology, where membrane fusion is important in various different cell processes. Although direct observation of the fusion process is not yet possible, the stalk model [174] provides a plausible mechanism, in which the cis membranes in the two bilayers first form a stalk, and then the trans layers make contact to fill the void created as the stalk increases in size. Finally, the contact area between the trans membranes increases to form a diaphragm, and a pore between the two membranes eventually forms. This process was investigated at the mesoscale in [154] in which the authors were able to simulate the fusion of two membranes. However, because fusion occurs over such a large timescale, it was necessary to introduce an artificial tension at several stages of the process to allow the fusion to occur.

It has been shown experimentally that properties of fluid bilayers such as the area stretch modulus and bending rigidity are determined by the structure of the component amphiphiles. In [133], DPD simulations were performed of bilayers composed of double tail amphiphiles with symmetric and asymmetric tails. It was found in the symmetric case that for a constant head-head bead interaction parameter, the area stretch modulus of the membrane increases as the size of the tails increases. The authors were able to determine a relation between the tail length and head size that resulted in an area stretch modulus that was independent of tail length for amphiphiles with 3 or 4 head beads. The authors were able to reproduce the experimental findings for the independence of the area stretch modulus on lipid tail length if the head-head interaction parameter was altered accordingly for the given tail length.

Shillcock et al. [233] present a bilayer model involving parameters which differ slightly from the standard DPD model in that the maximum dissipative force parameter between beads is specified between each type of bead and chosen to ensure the formation of a well ordered bilayer, rather than being uniform throughout the simulation. In this work, the authors simulated bilayer composed of single and double tail amphiphiles, which were not specifically mapped to real amphiphiles, and determined the lateral stress distribution across the bilayers. The results were compared with those from previous MD simulations and were found to be in good agreement except for double tail amphiphiles where the lateral stress distribution lacked increases in tension at the centre of the membrane. The authors suggest that this difference is due to the lack of a chain stiffness potential.

A very interesting discussion of the affects of coarse graining on the properties of dimyristoylphosphatidylcholine (DMPC) based bilayers is given by Kranenburg et al [143], building on their previous work [146] on phases and interdigitation in bilayers. The authors define a formula to convert interaction parameters between different mappings, but show that the relation between sets of interaction parameters and the level of coarse graining (e.g. the number of water molecules represented by a single DPD bead, assuming uniform bead density) is not linear. A comparison of a 1:1 mapping of water molecules to DPD beads, a 3:1 mapping and molecular dynamics simulations is performed with several different interaction parameters sets from previous works, showing that the the 3:1 mapping is actually much better than the 1:1 mapping, due to effects of the conservative force being somewhat unrealistic at atomistic scales. The results of this paper illustrate that the

mapping of DPD potentials to real world potentials is not trivial.

The action of proteins on membranes is an active area of research, and DPD has been used to investigate the result of actions such as cleaving of amphiphiles by proteins such as phospholipase (PLA_2, a membrane protein which digests amphiphiles) on bilayer properties [136]. Although the action of the protein could not be explicitly simulated, the results of amphiphile cleaving could be approximated by breaking bonds in a randomly selected subset of the bilayer amphiphiles. Results indicated that the hydrolysation of the amphiphiles produces a marked decrease in both the bending rigidity and area compression ratio of the membrane.

5.2.2 Simulation of Vesicles

Vesicles are composed of bilayer membranes that have formed spherical fully closed structures, thus delimiting a volume of solution from the surrounding solvent environment. Vesicles can be categorised by size as "Small" ($20nm$ to $50nm$ in diameter), "Large" ($100nm$ to $1000nm$ in diameter) or "Giant" ($1\mu m$ to $300\mu m$ in diameter). In biology, vesicles composed of phospholipid bilayers are produced in eukaryotic cells and act as containers in processes such as exocytosis and endocytosis. In some respects a vesicle can be seen as a model cell membrane, as they are typically composed of the same amphiphilic molecules, but are generally uniform in composition, unlike cell membranes which contain embedded proteins and molecules that perform a number of different functions such as membrane poration and inter-cell communication. For this reason vesicles are likely to be integral to protocell assembly [198, 253] and are also being investigated as possible vehicles for drug delivery [51]. Vesicles formed from non-lipid based amphiphiles (typically diblock co-polymers) are termed "polymersomes" and generally have stronger membranes [74] than lipid based vesicles ("liposomes").

Although there are several pathways for the formation of vesicles (see Figure 5.1), the most interesting is perhaps the spontaneous self-assembly of vesicles from a solution of amphiphiles. This process occurs at length and time scales that make direct observation intractable. However, simulation techniques have been useful in investigating the formation process and some efforts have been made towards vesicles simulations using coarse grained molecular dynamics [176, 175], Brownian dynamics [195, 196, 197, 194] and even atomistic molecular dynamics [71]. Despite the reduced force interaction radius, coarse grained molecular dynamics is computationally very expensive and so solvent particles are not explicitly simulated in the above references, instead being replaced with local density based hydrophobic potentials.

DPD was first shown to be a good model for simulation of vesicle formation by Yamamoto and Hyodo in 2002 [272]. In this paper the authors create a set of parameters that model a number of different amphiphiles' structures although the amphiphiles were not specifically mapped to any real amphiphilic molecule. The interaction parameters between the head, tail and solvent beads were then chosen to represent a strong segregation seen in di block copolymer systems. The authors

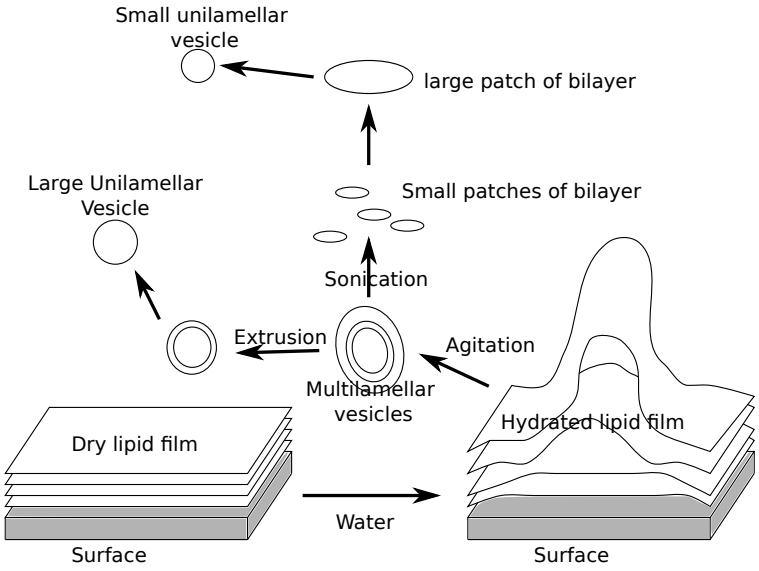


Figure 5.1: The different pathways for the formation of small and large unilamellar vesicles

were able to demonstrate vesicle formation simulations using the DPD model with these parameters.

The results of these simulations are illustrative of the vesicle formation pathway from an initial state of amphiphiles randomly dispersed in solvent, which has also been observed in coarse grained molecular dynamics simulations [176]. The fact that this process can be observed by simulating with different methods and parameterisations is indicative of its mesoscopic nature. The vesicle formation occurs due to the net effect of the underlying atomic interactions rather than due to the specifics of the atomic interactions themselves.

In a more recent simulation, the same authors then investigated the budding and fission of domains in vesicles composed of more than one type of amphiphile [270]. Indeed, budding and domain formation in vesicles has been well studied using DPD with investigations performed into the effect of membrane tension and interaction parameters on domain growth, budding and fission of vesicles composed of two types of amphiphiles [150, 151, 152]. The effect of area per molecule and tail length on phase separation in vesicles and bilayers was investigated in [134].

Simulations of di-block co-polymers by Ortiz et al. in [204] are notable due to a modified mechanism for mapping the DPD representation of an amphiphile to the real molecule. It was found that when using the standard method for mapping of moieties to DPD particles based on the density of water molecules, described in [111] an unrealistic density profile was observed for the hydrophobic core due to the mapping of three water molecules per DPD bead being used to define the density of all bead types. The proposed solution was to introduce a new density mapping that allows bead densities to differ between the different types, chosen to more accurately reflect the experimental values for the bulk density of the pure species.

In the living cell, fusion of vesicles is an important process in the transport of signalling

chemicals. In the synapse for example, neurotransmitter is released into the synaptic cleft by a process of vesicle fusion with the membrane of the axon. The mechanism by which vesicles fuse at the molecular scale with other membrane structures has not yet been directly observed due to the short length and time scales involved, although some progress has been made with giant unilamenar vesicles [118]. A theoretical model describing membrane fusion has been proposed in the form of the stalk model [174, 55] which was investigated in a DPD simulation by Li et al. [154]. Membrane fusion in cells is a complex process involving molecules that tether the two membranes together and membrane associated proteins to produce favourable tensions for fusion to occur [55]. Simulation of vesicle fusion, which occurs over longer timescales that most other processes that have been simulated with DPD has become feasible with today's hardware, and a systematic investigation of vesicle fusion with a bilayer membrane was performed in [232]. Although no mapping was performed between the DPD polymers and any real amphiphilic molecule, the authors were able to consistently show fusion between a small vesicle and a patch of bilayer when tension was applied to the vesicle and membrane bilayers. This work was further extended in [234] where the authors simulate vesicle-membrane fusion assisted by membrane embedded SNARE proteins, modelled in DPD by creating cylinders of amphiphiles in the bilayer that were immobilised as one large solid inclusion.

5.2.3 Simulations of Protocells

DPD has already been used for several investigations into the general behaviours of protocell models. The simulation of protocells is relevent to the the modelling of vesicle computing systems, as model protocells are typically composed of membranes encapsulating simple reaction systems. Simulations by Fellermann et al. [86] involve the study of an information free protocell scenario of replicating micellar containers. In this protocell model, each protocell is composed of a micellar container produced by the self-assembly of amphiphiles, represented with simple hydrophillic/hydrophobic dimers similar to those in the Jury model. Hydrophobic precursor dimers are introduced into the system as a constant rate, and a transformation reaction occurs which changes the precursor into an amphiphile, but only in the presence of other amphiphile dimers (the amphiphiles are autocatalytic). As the transformation of precursors to amphiphiles was localised by the catalytic effect of the existing monomer, the amphiphile resulting from the reaction would typically insert into the aggregate containing the catalysing amphiphile, thereby increasing the size of the micelle aggregate. When the micelles reach a critical size, the aggregate becomes unstable and divides into two daughter micelles, each capable of catalysing the production of amphiphiles. Simulations were performed in DPD, and the protocells were shown to increase in size and divide over the course of the simulation. Although this model showed a very simple method by which protocells could self-replicate, the authors intended not to include information transfer between protocell generations.

Later work by Fellermann et al. [87] with an extended DPD simulation, investigates another protocell model which has micelles as the container, but augments the information free model above

with small oligomers representing peptide nucleic acid (PNA) chains with hydrophobic anchors which embed on the surface of the micellar protocells. This model protocell, named the "Los Alamos bug" is similar to the replicating micelles described in the previous paragraph. However, the catalysis of amphiphile formation is the result of the PNA gene rather than being due to amphiphilic autocatalysis, and information is also included in the model by the introduction of the PNA chains, which replicate by template directed ligation. As the sensitizer molecules are hydrophobic, they will tend to embed within the lipid core of the micelles, causing the micelle to catalyse the production of new amphiphiles. The sensitizer molecule also catalyses the elongation of the replicating PNA chain. The protocells will grow as more amphiphiles are produced, and the PNA chain will be replicated on the surface. When the size of the protocell reaches the critical size it will divide into daughter cells which should both contain the PNA chain. In this work, the authors use DPD to simulate the life cycle of the protocells described in this model.

5.3 An Object Library for Dissipative Particle Dynamics

To allow the combinatorial exploration of liposome logic using a variety of vesicles, with different properties and composed of different molecular species, functionality was added to the DPD toolkit to allow the extraction, manipulation and storage of self-assembled structures. By adding this functionality, the modeller is able to construct more elaborate models, and explore membrane configurations that may not be very likely to occur during the usual self-assembly process. This allows the removal of any extraneous self assembled structures which are irrelevent to the system being modelled. By locating and analysing objects in simulation data it becomes possible to modify the contents of the vesicle volume and to introduce reactants etc.

For example, a new initial environment in which a large vesicle is modified so that some of the solvent particles in the inner core have there types changed to those of chemical reactants in the BZ reaction could be created by selecting a vesicle from the vesicle library and creating a simple configuration file which specifies the modified contents of the vesicle. This initial state could then be simulated to study the encapsulated dynamics of such a reaction.

5.3.1 Creation of the Object Library

The object library was created by simulating the formation of vesicles using the DPD toolkit, using a combination of the serial, parallel and CUDA implementations of the DPD technique which were described in detail in the previous chapter. A number of different amphiphile parameter sets were used for vesicle formation, and then the data files from the vesicle formation simulations were analysed using the *dpdanalyser* program in the DPD toolkit, which was used to identify and extract the assembled vesicle objects which are then stored as data files in the vesicle library. For model amphiphiles which may not self-assemble within the timescales of a DPD simulation, the *dpdtimestep* tool was used to create an initial state for a simulation in which the amphiphiles and solvent were

arranged to form a bilayer vesicle.

In order to maximise the predictive power of the models, only those amphiphile parameter sets in the literature that were derived from real amphiphiles, either by calibration with experimental data or more detailed molecular dynamics simulations were chosen. Of those parameter sets which were suitable, two different model amphiphiles were chosen. the first is based on the DMPC phospholipid, and the second is a model of the PEE-PEO diblock copolymer. These two molecules are representative of two important classes of amphiphiles, phospholipids which are commonly found in biological membranes, and copolymers which are commonly used for the creation of synthetic polymersomes, which are of great interest in fields like pharmacy and bioremediation.

5.3.2 DMPC vesicles

For the model DMPC amphiphiles, the parameter set detailed in [144] was used. This parameterisation is based on the coarse graining of three water molecules per DPD bead with a number density $\rho = 3$. Figure 5.2 shows the mapping of the DMPC molecule to DPD head and tail beads.

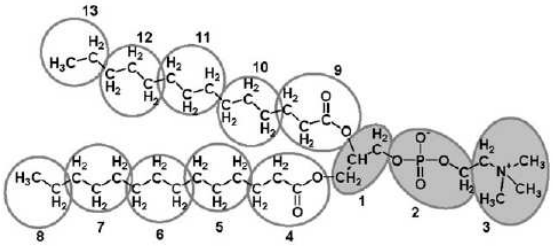


Figure 5.2: The coarse graining of the DMPC amphiphile into DPD beads with volume equivalent to three water molecules ([144] - Reproduced by permission of the PCCP Owner Societies). The empty ovals represent the mapping of the hydrophobic tail DPD beads to the molecular structure, and the grey filled ovals represent the mapping of the hydrophilic head DPD beads to the molecule.

In terms of the structure of the amphiphile, the bonds between particles were defined so that all bonds have a preferred length of $r_0 = 0.7r_c$ and the spring force parameter is set to $k = 100k_bT$. Harmonic bond angle forces were imposed along the amphiphile tail chains and at the join between the two amphiphile tails and the head group. No angle forces were imposed on the headgroup particles, and so these were able to move freely. The parameters for the angle forces are as follows, the preferred angle for the tail chains was $\theta_0 = \pi$ with a maximum force $k_\theta = 6$, and the parameters for the angle potential between the bonds connecting the tail chains to the headgroup, was a preferred angle of $\theta_0 = \frac{\pi}{2}$ and a maximum force $k_\theta = 3$. The conservative force parameters for the interactions between different particle types are shown in Table 5.1.

To simulate vesicle formation using the model DMPC amphiphiles, the amphiphiles were placed randomly in a cubic volume of size $50r_c^3$ for 10000 time units. The concentration was such that the volume fraction of DMPC molecules was between 0.18 and 0.19 with these values being determined empirically to produce vesicles. If the volume fraction was much lower than 0.18 then the

Type	Water	Tail	Head
Water	78	104	75.8
Tail	104	78	104
Head	75.8	104	86.7

Table 5.1: The conservative force parameters for interactions between the different particle types in the DPD model of the DMPC amphiphile.

resulting micelles would not form a large enough oblate micelle to produce a vesicle, and if the volume fraction was greater than 0.19 then bilayers would form across one of the planes of the simulation space. Vesicles were found to form after roughly 5000 time units, Figure 5.3 shows the pathway of the vesicle formation in the simulations. Initially the amphiphiles are distributed randomly, and quickly form small micelles. These micelles then join together to form larger micelles, until a large oblate micellar patch of bilayer is formed, which then begins to curve upwards or downwards to form a bowl shape. The bowl shaped bilayer will eventually completely close over, forming the spherical vesicle.

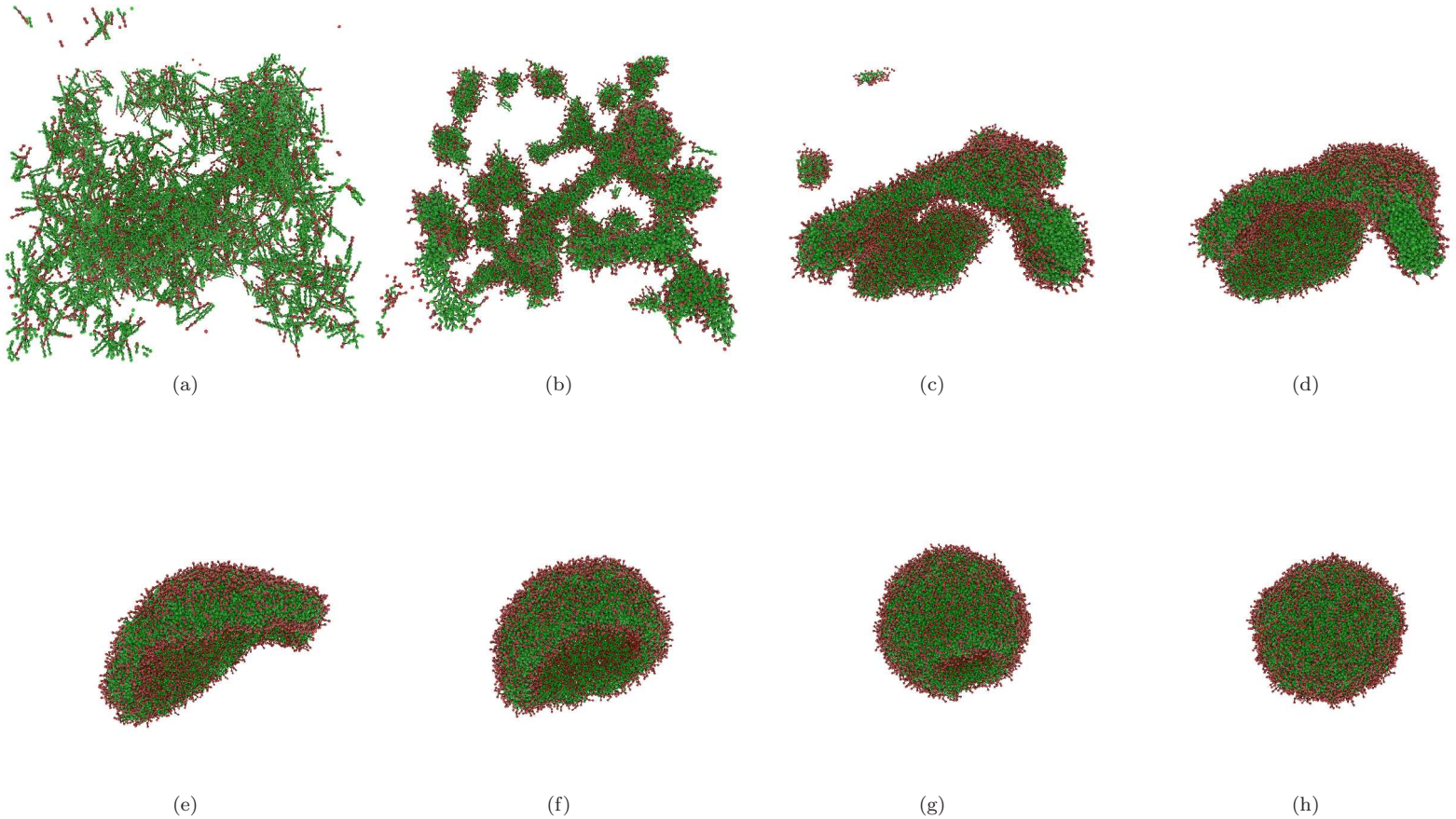


Figure 5.3: The process of vesicle formation from an initially random configuration of amphiphiles (a). The amphiphiles first come together to form micelles (b) which accumulate and the micelle becomes more oblate (c) and (d). If the micelle is large enough, the edges then start to curl (e) until the micelle becomes bowl shaped (f). Eventually, the bowl shape closes over (g) and a vesicle if formed (h). Since the process takes a large amount of simulation time to occur, it is current practice in the literature when studying the properties of vesicles to start simulating from a pre-assembled configuration of a closed or nearly closed vesicle.

A large scale analysis of the formation of DMPC vesicle was performed using the DPD toolkit by performing 100 simulations of vesicle formation in a volume of $50r_c^3$ and 10 simulations of the same volume fraction of DMPC performed in a volume of $90r_c^3$. The simulations were performed using the CUDA implementation, with each run taking approximately 2 hours to complete. Extracting and storing these vesicles enables an analysis to be performed across the whole vesicle library. The results of this analysis are now discussed.

The simulations of DMPC vesicle formation resulted in the formation of 163 vesicles and 1750 micelles. Using *dpdanalyser* the data from the DPD simulation can be analysed to identify any membrane objects in the simulation space, recording information about the objects, such as the number of amphiphiles in the object, and information the Minkowski functionals in an XML file. This data is then analysed to characterise the properties of the vesicles which form in the DPD simulations. Firstly, it is interesting to know how many amphiphiles compose each vesicle or micelle. Figure 5.4 shows the distribution of the number of polymers in vesicles and micelles for the objects identified in the object library simulations. The figure shows that vesicles are unlikely to form from

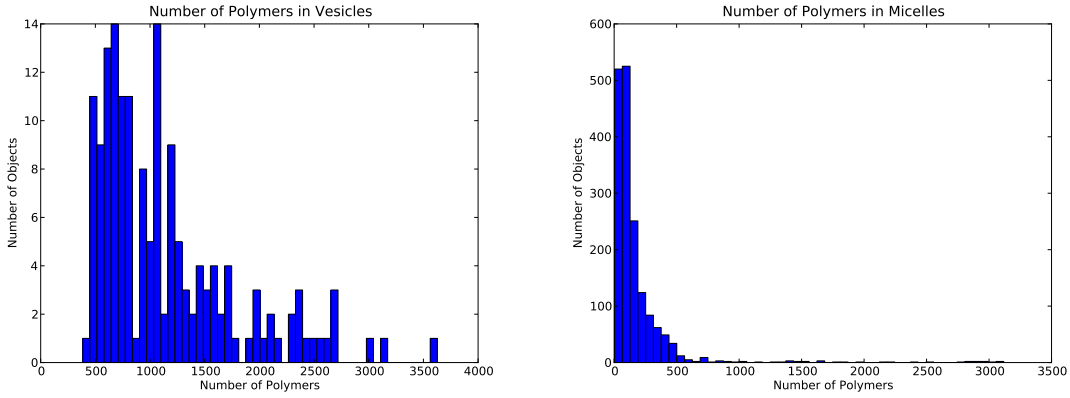


Figure 5.4: Histograms showing the size distribution of the vesicle and micelle objects extracted from simulation data

less than 500 polymers, with the most likely range of polymers for vesicles being between 500 and 1750, although vesicles containing over 3500 polymers were formed. The distribution for the number of polymers found in micelles on the right of Figure 5.4 shows that micelles most likely contain less than 500 amphiphiles, although some micelles contained over 3000 amphiphiles. This result supports the interpretation that once the micelle reaches around 500 amphiphiles during the self-assembly process, it will be large enough to form a vesicle. The relationship between the number of polymers in the vesicle or micelle and the surface area is shown in Figure 5.5. The figure shows a very similar relationship between the number of amphiphile in vesicles and micelles and the surface area. This indicates that the contribution to the surface area from each amphiphile is very similar, regardless of whether the amphiphile is located within a vesicle or micelle. The relationship between the number of polymers in the vesicle or micelle and the curvature, calculated as one of the Minkowski

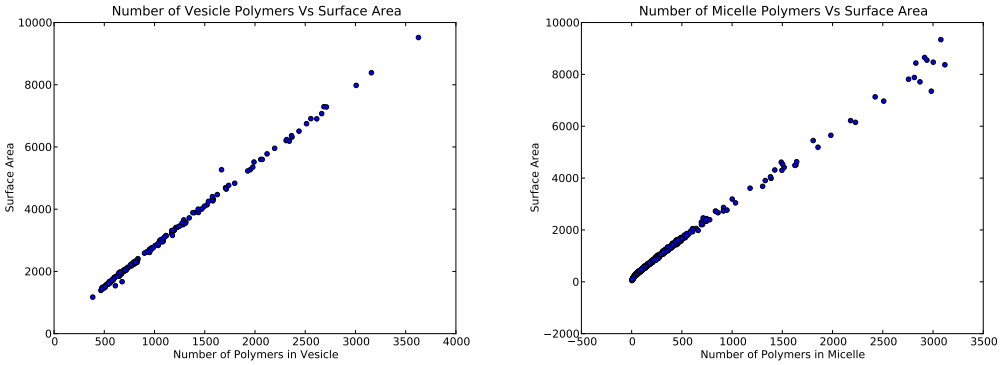


Figure 5.5: Scatter plots showing the relationship between the number of amphiphiles in the object and the object surface area

functionals, is shown in Figure 5.6. The figure shows that for vesicles, the curvature does not vary

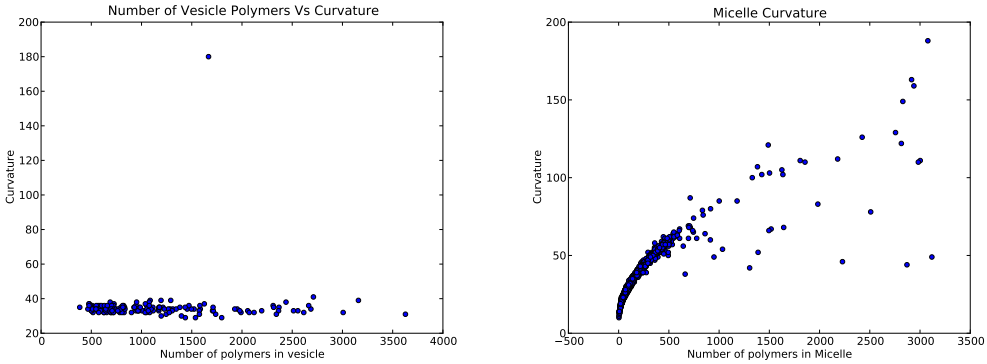


Figure 5.6: The relationship between the number of polymers and curvature for both vesicles and micelles. Note that the data point for one bilayer object containing roughly 1750 amphiphiles has a much greater curvature value than the curvature values of the other vesicles. This anomalous value is due to the mischaracterisation of this bilayer object as a vesicle by the vesicle identification algorithm.

greatly with the number of polymers in the vesicle, and therefore the vesicle size.

Figure 5.7 shows the relationship between the number of polymers in a vesicle, and the number of solvent particles which are encapsulated within the vesicle.

5.3.3 Di-Block Copolymer Vesicles

The model di-block copolymers vesicles were created using a model amphiphile proposed in [204] and derived from molecular dynamics simulations to produce a good agreement with MD simulations of the same amphiphile in terms of the bilayer properties. The structure of these amphiphiles is shown in figure 5.8. The parameters for the model amphiphile were derived by attempting to calibrate the dynamics of the DPD polymer with those in MD simulations. For the bonding potentials between

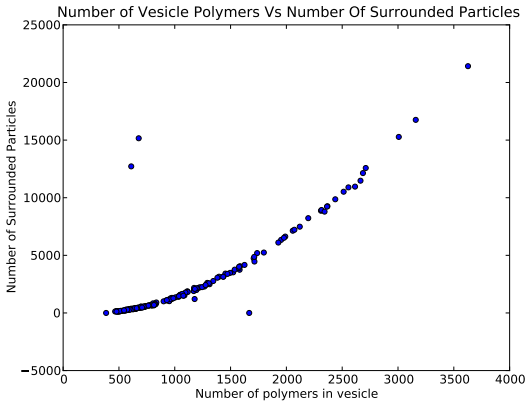


Figure 5.7: The relationship between the number of particles encapsulated within the vesicle and the number of polymers in the vesicle membrane. The outlying values are a result of the mischaracterisation of bilayer objects as vesicles by the vesicle identification algorithm.

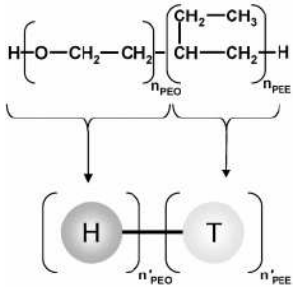


Figure 5.8: The coarse grained mapping between the PEO-PEE copolymer and the DPD beads in the simulation (reprinted ("Adapted" or "in part") with permission from [204]. Copyright 2005 American Chemical Society.) In this mapping, just as for the DMPC amphiphile, each DPD bead has a volume equivalent to three water molecules.

particles in the polymer chains, the parameters are shown in table 5.2. The parameters for the angle potentials in the polymer chains are shown in table 5.3. The conservative force interactions parameters between the different bead types are shown in table 5.4:

As the di-block copolymer vesicles do not generally self-assemble, and are instead typically formed by other pathways such as sonication of multilamellar bilayers or hydration, PEO-PEE vesicles were formed by creating an artificial configuration of amphiphiles, such that the amphiphiles were arranged to form a vesicle membrane according using the *dpdtimestep* program according to the specifications provided in [204]. In this paper, the authors generate PEO-PEE vesicles by evenly distributing 1569 amphiphiles, which had a hydrophilic PEO block 40 monomers long, and a hydrophobic PEE block which was 26 monomers long, over the two membrane leaflets such that the area per molecule was $3.75nm^2$. This resulted in the outer layer containing 1255 amphiphiles, and the inner layer containing 314 amphiphiles. Reproducing this configuration using the DPD toolkit software, and simulating the vesicle for 2000 time units with a step length of $\delta t = 0.02$ showed that

Bond	r_0 (r_c)	k (ϵ)
H-H	0.5573	1107.51
H-T	0.64969	566.84
T-T	0.5998	160.93

Table 5.2: The bond potential parameters for Head-Head, Head-Tail and Tail-Tail bonds.

Angle	θ_0	k_θ
H-H-H	122.1	4.03
H-H-T	145.5	4.96
H-T-T	97.8	13.48
T-T-T	102.2	10.19

Table 5.3: parameters for the bond angle potentials for angles between the different particle types.

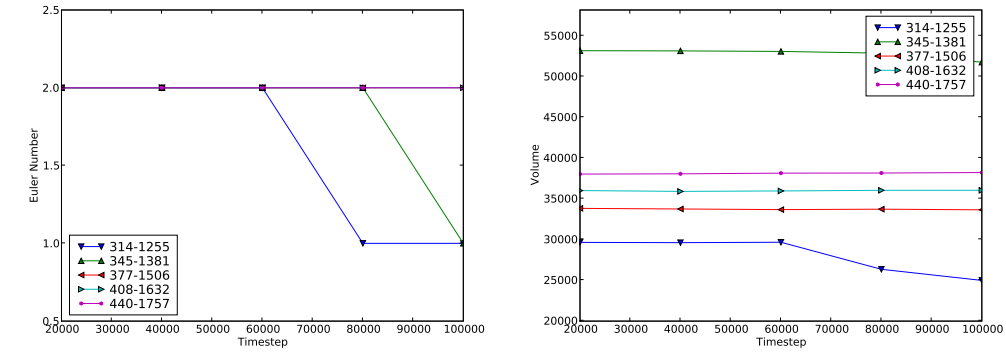
the vesicle did not remain stable. Simulations of this configuration were performed within a volume of $50r_c^3$ for 8000 time units to allow the vesicle structure to equilibriate. It was found that placing the number of amphiphiles in the inner and outer leaflets which were stated in [204] resulted in vesicles which did not remain stable over the course of the simulation, and burst after roughly 4000 time units. It was hypothesised that this could be due to a disparity between the manner in which the polymers were placed in this study and in the Ortiz et al. study resulting in the creation of a membrane which was under too much tension, as the method which the authors used to place the amphiphiles was not specified in the paper. To determine if this was the case, the number of polymers in the inner and outer vesicle membrane leaflets were increased by 50% in increments of 10%. Details of the Minkowski analysis of these simulations are shown in figure 5.9. The change in Euler number in 5.9a from 2 to 1 indicates that the vesicle has burst. When the polymer increase was 0% or 10% (green and blue lines) the resulting vesicle was found to burst. However, increasing the number of polymers in the membrane by 20% or more resulted in a vesicle which was stable for the duration of the simulation.

5.4 A new model PEO-PLA amphiphile

In this section, the development of a new set of model parameters for a diblock copolymer amphiphile composed of polyethylene oxide (PEO) and polylactic acid (PLA) monomers is presented. The model

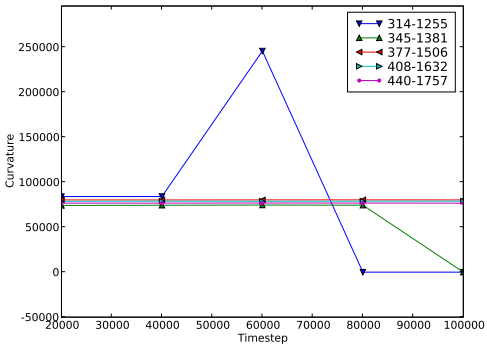
Type	Water	Tail	Head
Water	78	100	79.3
Tail	100	78	86.7
Head	79.3	86.7	78

Table 5.4: The conservative force parameter for interactions between the different particle types

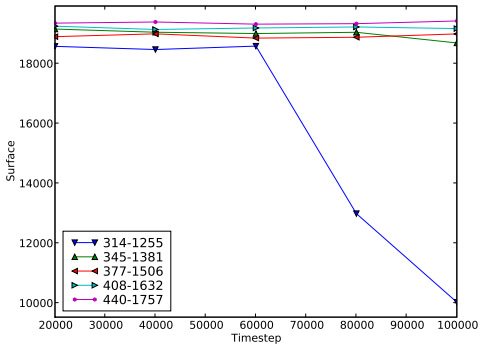


(a)

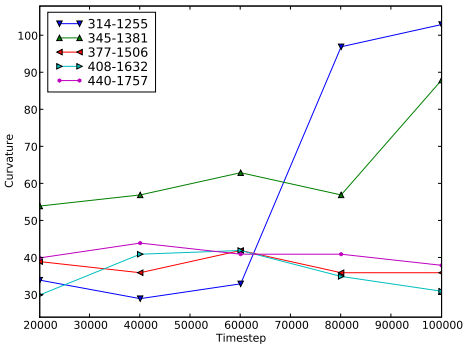
(b)



(c)



(d)



(e)

Figure 5.9: The Minkowksi functionals based analysis of the diblock copolymer vesicles, created using the model amphiphiles proposed in [204]. The Euler number (a), volume (b), number of particles surrounded by the vesicle (c), the vesicle surface area (d) and the vesicle Gaussian curvature (e) different numbers of polymers in the inner and outer leaflets of the vesicle are shown. In each figure the blue line shows the vesicle with the same number of polymers in each leaflet as specified in [204], and the remaining lines show vesicles where the membranes were increasingly more dense.

was developed in collaboration with an experimental investigation into the formation of PEO-PLA vesicles conducted at the University of Nottingham School of Pharmacy.

Polymersomes formed from PEO-PLA amphiphiles are of interest in pharmacy as the polymersome membranes tend to be porated. By tuning the length of the blocks in the copolymer, it may be possible to control the level of poration and therefore the rate at which encapsulants would leak from the vesicle. This property could also be of interest in vesicle computing, as self-porating polymers may permit increased diffusion across the membrane without the need to express pore proteins within the polymersome core. By creating a well calibrated DPD model of the amphiphile it would be possible to rapidly gain insight into the effect of modifying the lengths of the polymer chains in the copolymer on the properties of the polymersome.

A number of studies have been published which present models of the different PEO and PLA blocks, as was previously mentioned. A model PEO-PEE amphiphile for the modelling of vesicle membranes was presented in [204] and [167] present a DPD model of PEO and PVC (polyvinyl chloride) polymer blends, and [153] showed a model of polyethylene (PEE) and PLA polymer blends and diblock copolymers. DPD was used to study polymer blends of micelles composed of PEE-PEO and homopolymer polypropylene oxide in [274] and [157] studied the microphase separation of polystyrene-isoprene diblock copolymers. A model of the formation of PLA microspheres was constructed in [116] and a triblock copolymer model of PEO-PPO-PEO block copolymer micelles, and their role in the stabilisation of gold nanoparticles, was investigated in [54]. In [117] the authors present a model PEO-PLA amphiphile in DPD, to simulate the formation of the self-assembling of a self-assembled fibre like structure, which was impregnated with Paclitaxel, a drug molecule. Table 5.5 provides a summary of these references and the different parameters which were specified for the PEO and PLA block models.

Paper	Type	water mapping	PEO Mapping	PLA Mapping	a_{EO-W}	a_{LA-W}	a_{EO-LA}	bond len/k
Chen 2007	hydrated	1	4 ($240\AA^3$)	-	35.93	-	-	0.0/4.0
Luo 2009	blend	-	4.98	-	-	-	-	0.0/4.0
Lee 2007	blend	-	-	2 ($191\AA^3$)	-	-	-	0.0/4.0
Zhao 2009	hydrated	1 (α) 20 (vol)	9 ($600\AA^3$)	-	26.05	-	-	0.0/4.0
Guo 2002	hydrated	$150\AA^3$	$150\AA^3$	-	0.3 (χ)	-	-	-
Guo 2006	hydrated	1	-	1	-	69.62	-	0.0/4.0
Guo 2007	hydrated	1	-	1	-	69.62	-	0.0/4.0
Guo 2009	hydrated	1	1 ($190\AA^3$)	2 ($190\AA^3$)	36.92	71.37	33.24	0.0/4.0
Ortiz 2005	hydrated	3	1.392	-	79.3	-	-	see paper

Table 5.5: Table of PEO and PLA parameterisations, the *Type* of model is either hydrated (simulated in water) or blend (the polymers are simulated as blends without solvent). The three *Mapping* columns specify the bead number N_m for water, PEO and PLA particles in each model. The a parameters show the conservative force interaction parameters between particles of PEO and water (EO-W), PLA and water (LA-W) and PEO and PLA particles (EO-LA) and the *bond len/k* gives information regarding the preferred length and strength parameters for the bond forces.

As a model of PEO-PLA diblock copolymers is presented in [117], the first stage of this work involved the reproduction of the Guo model amphiphiles, to determine if they could be used for vesicle formation. A configuration file for the DPD simulator was created that reproduced the parameters presented in the paper, and amphiphiles with a structure $EO_{50} - LA_{50}$ were created. Since each

Area Per Molecule (r_c^2)	Inner Polymers	Outer Polymers
4	1964	3849
5	1571	3079
6	1309	2566
7	1122	2199
8	982	1924
9	873	1710
10	785	1539
11	714	1399
12	654	1283
13	604	1183
14	561	1100
15	524	1026
16	491	962

Table 5.6: The number of polymers in the inner and outer leaflets of the initial polymersome configurations

PLA bead in the Guo et al. DPD model maps to two PLA monomers, the resulting amphiphiles were composed of a block of 50 PEO beads, connected to a block of 25 PLA beads. Polymersomes composed of this model amphiphile were created using the artificial vesicle creation method described in the previous section, after simulating a single polymer to determine its equilibrium structure. Different artificial vesicle configurations were then created to form vesicles which had initial diameters of roughly $60r_c$, with amphiphiles placed in the inner and outer layers such that the area per polymer ranged from between $4r_c^2$ to $16r_c^2$. The membrane thickness was assumed to be $10r_c$. Table 5.6 shows the number of polymers placed in the inner and outer layers for the different areas per molecule.

Each vesicle was equilibriated for 50τ, and then simulated for a further 20000τ, and particle position and velocity data was recorded every 2000τ for later analysis with the *dpdanalysis* tool. The results of the simulations indicated that the initial vesicle configurations all degenerated into large micelles. Figure 5.10 shows an example from the results of the simulations, showing the initial vesicle configuration, and the resulting micelle structure in cross section and in profile.

There are several aspects of the Guo model amphiphiles which may be described too coarsely for successful vesicle simulation. Firstly, the coarse graining of the water, PEO and PLA molecules into DPD beads implies that a PLA bead has a volume of roughly $190\mathring{A}^3$, but the water in the system is coarse grained such that a water bead in the simulation represents a single molecule of water (with volume of roughly $30\mathring{A}^3$). The volume of the beads in DPD should be equal, and so this disparity results in a softening of the interactions between the di-block copolymers and the surrounding water particles. Secondly, the intra-molecular interactions specified by the Guo model were configured such that the bond parameters, which determine the preferred bond distance and maximum strength of the harmonic bond force, produce a bond distance which is most probable according to the radial distribution function of the ideal DPD fluid. Setting the bond parameters

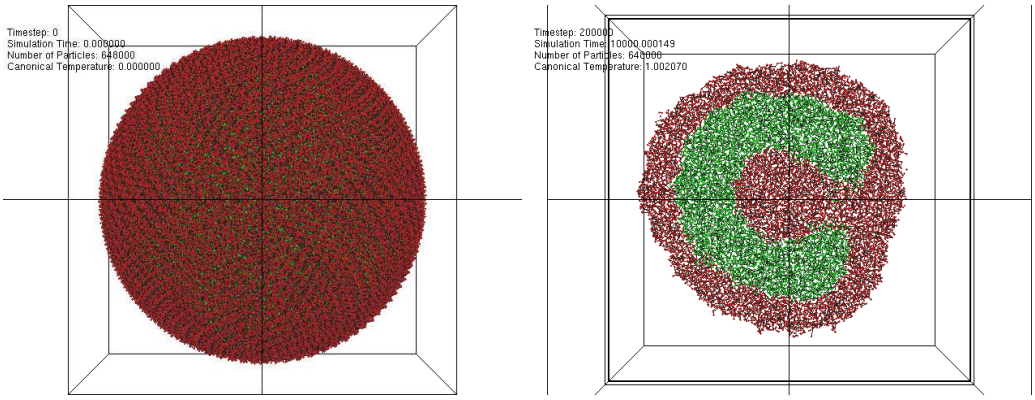


Figure 5.10: Simulation of an artificial vesicle configuration, the initial configuration (left), and the resulting micelle in cross section (right).

in this way does not represent the structural properties of the di-block copolymer. This problem is compounded by the lack of bend angle forces in the model which means no stiffness is imposed in the polymer chain. It is likely that the correct setting of these parameters is of greater importance for the simulation of vesicles than it is for the simulation of micelles (either spherical or cylindrical), which was the original application for the Guo et al. model amphiphiles. Also, the parameters proposed by Guo et al. report a fairly high conservative force interaction parameter for the interaction between the PEO monomers and water. This seems unlikely, as PEO is found to dissolve in water at almost any concentration [204].

In order to address these issues with the original PEO-PLA model a new PEO-PLA model amphiphile was created, with the focus placed on attempting to accurately specify the bond parameters between beads, and to reduce the strength of repulsion between the PEO-Water interactions. The new parameterisation was based on χ parameters from the [117] and [204] and molecular dynamics simulations of the PEO-PLA polymer to determine the bond angles. Table 5.7 shows the a parameters for the conservative force interactions between water, PEO and PLA particles that were defined in [117].

Type	Water	PEO	PLA
Water	25	36.92	71.37
PEO	36.92	25	33.24
PLA	71.37	33.24	25

Table 5.7: The a conservative force interaction parameter matrix from the Guo et al. 2009 model

The relationship between the Flory-Huggins χ parameter and the DPD a parameter is $\frac{a_{ij}-a_{ii}}{3.27} = \chi$, where in this case $a_{ii} = 25$. The χ parameters for the model are shown in table 5.8. Note that as the PLA bead represents two monomers in the original model, the resulting χ parameter is halved to get the value for a single bead.

The mapping in the new model will be set such that a DPD water bead represents a volume

Type	Water	PEO	PLA
Water	0	3.65	7.09
PEO	3.65	0	1.26
PLA	7.09	1.26	0

Table 5.8: χ parameters for PLA-PEO in the Guo et al. 2009 model

of three water molecules ($90\mathring{A}^3$), with a PEO bead representing a single PEO monomer, and a PLA bead representing a single PLA monomer. Despite this mapping meaning that the PEO and PLA beads have a slightly larger volume than the water beads, the disparity is not as great as in the previous model, and the selection of this mapping enabled the selection of the PEO-Water interaction parameter from the [204] paper. Table 5.9 shows the a parameter matrix for the interactions between PEO, PLA and water particles in the new model.

Type	Water	PEO	PLA
Water	78	79.3	101.2
PEO	79.3	78	82.1
PLA	101.2	82.1	78

Table 5.9: The a conservative force parameter interaction matrix for the new PEO-PLA copolymer amphiphile model

The remaining aspect of the new model is the derivation of a set of bond and angle parameters for the bond and angle forces acting between beads. To determine these parameters, the process specified in [204] was followed. To derive the forces, molecular dynamics simulations of the polymer molecule are performed, and the positions of the atoms in DPD are averaged to determine a centre of mass position for the atoms in MD which make up each bead in DPD (in this case the positions of the atoms which compose each monomer are averaged). These centre of mass positions are then tracked throughout MD simulations of the polymer and the average and standard deviation of the distances and angles between bonds are calculated. The bond parameters in the DPD model are then set using trial and error such that the mean and standard deviation of the observed bond distances and angles resemble as closely as possible the values from the MD simulations. The molecular dynamics simulation package *Gromacs* [159] was used to simulate a small instance of the PEO-PLA diblock copolymer, with 5 monomers of each block. Table 5.10 lists the new bond parameters, and Table 5.11 shows the angle parameters derived from this method.

The artificial vesicle experiments listed in table 5.6 were repeated using the new model, and the vesicles were shown to remain stable for 20000τ. Figure 5.11 shows an example stable vesicle.

5.5 Summary

In this chapter the construction of the library of membrane objects is described. The intention of the library is to store membrane modules, similar to the biobricks repository in synthetic biology,

Types	Preferred Bond Length (r_c)	Bond Force Strength
PEO-PEO	0.5127	1400
PEO-PLA	0.6537	980
PLA-PLA	0.5472	1380

Table 5.10: The bond parameters for the new model PEO-PLA amphiphiles

Types	Preferred Bond Angle (Radians)	Angle Force Strength
PEO-PEO-PEO	2.3618	12.0
PEO-PEO-PLA	1.8788	11.0
PEO-PLA-PLA	1.4565	30.0
PLA-PLA-PLA	1.6164	21.0

Table 5.11: The angle parameters for the new model PEO-PLA amphiphiles

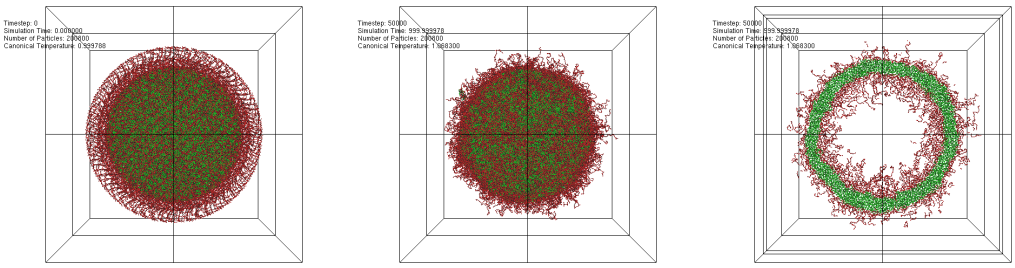


Figure 5.11: The result of simulating an artificial vesicle configuration using the newly derived model PEO-PLA amphiphile. The initial configuration (left), relaxed into a stable vesicle, shown in profile (centre) and in cross section (right).

which eases the process of modelling vesicle computation and other vesicle systems by allowing the model to be created from the pre-simulated objects from the library. In combination with the other applications in the DPD toolkit, such as the *dpdtimestep* command, the vesicles in the library can be used to create initial simulation states. As well as membrane objects and vesicles encapsulating solvent, the vesicle library will also include vesicles containing chemical reactions necessary to represent Boolean logic functions, and the creation of these vesicles will be described in the next chapter. Adding these functionalised vesicles should enable the designer of the vesicle computing system to quickly create intricate logic gate based models of computation in vesicles at mesoscopic levels of detail.

This chapter also reports the analysis of the formation of over 100 vesicles for inclusion within the vesicle library, with special interest paid to the distribution of vesicle properties such as the number of polymers in the membrane and the Minkowski functional data such as the volume, surface area and curvature of the object. By understanding the distributions of these properties, it becomes possible to create DMPC vesicles by creating artificial configurations of amphiphiles, which have similar properties to those formed from the self-assembly process. The analysis showed characterised the relationships between these properties and vesicle size, which enables predictions to be made about the properties of large vesicles which it may not yet be feasible to form in simulation, and acts as a proof of principle for the large scale automated analysis of simulation data with the automated analysis methods provided by the DPD toolkit.

The next chapter details the investigation into the diffusion properties of vesicles which contain pore inclusions within the vesicle membrane as a simple means of communication between vesicle computing devices.

Chapter 6

A Study of Diffusion Through Porated Membranes

In this chapter stochastic P systems and the DPD toolkit are used to develop sophisticated simulations of mechanisms for communication in vesicle computing systems. The aims of the work presented in this chapter are two-fold, firstly to investigate some possible communication mechanisms for vesicle computing in simulation, focussing on chemical communication, and secondly to illustrate the utilisation of the vesicle computing simulation framework. Diffusion of particles from within a vesicle with a membrane containing simple pores was investigated in simulation, and a qualitative comparison of the results is made with those from a similar wet-lab experiment. The work described in this chapter was nominated for best paper in the GECCO 2008 conference.

6.1 Introduction

Now that the vesicle computing Simulation and Modelling framework has been described in Chapters 3 and 4, I consider the simulation and modelling of one aspect of a simple vesicle computing system. This chapter focuses on possible mechanisms for the communication of information between vesicles in a fluid environment. Communication between computing elements in the cellular computing paradigm allows the computing elements to work in concert to solve larger problems, and allows inputs and outputs to be presented to and read from a vesicle computing system. We can turn to biological cells as examples of how communication between cells is utilised. Recent advances in analytical biotechnology, computational biology, bioinformatics and microbiology are transforming our views of the complexity of biological systems, particularly the computations they perform (i.e. how information is processed, transmitted and stored) in order to survive, adapt and evolve in dynamic and sometimes hostile environments.

For example, several species of bacteria utilise a type of chemical messaging known as *Quorum Sensing* to coordinate activities across the whole bacterial colony. In quorum sensing, each bacterium in a colony will synthesise small auto inducer molecules at a basal rate. Autoinducers can diffuse through the bacterial membrane and so as the colony grows, the concentration of autoinducer in the surrounding environment will rise. The bacterium also contains receptors for the autoinducer

molecule and so once the concentration of autoinducer reaches a critical level, the molecule will bind to the receptor which alters the pathway of gene expression within the bacterium, causing the virulence factor to be synthesised, whilst simultaneously increasing the production rate of the autoinducer, leading to a positive feedback loop which means the entire colony all start producing large amounts of virulence factor over a short period of time. Quorum sensing was first observed in Vibrio Fischeri bacteria, in which the autoinducer switches on the production of the luciferase protein when the colony becomes large enough, causing the whole colony to glow, but the mechanism has been observed in other bacteria such as Pseudomonas Aeruginosa which utilise quorum sensing to coordinate the formation of a biofilm when the colony becomes large enough to overcome the hosts' defences.

Chemical signalling offers a possible means of controlling the computation in a population of vesicles, all immersed in the same fluid environment, as the mechanisms which drive quorum sensing could be used as the basis for a crude chemical signalling system between computational entities. In this way, chemical computations occurring in vesicles could be synchronised easily with the introduction of autoinducer molecules into the environment, either artificially, or as the output of another chemical computation occurring in other vesicles.

In the biological cell, proteins in the cell membrane allow the membrane to act as a filter, allowing the the cell to increase the concentration of beneficial substances within its cytoplasm, even against concentration gradients, whilst expelling waste products and other byproducts of cellular processes which may be harmful if allowed to accumulate within the cytoplasm. It is quite likely that a vesicle computer, implemented within the framework of a synthetic protocellular organism would require similar membrane functionality to survive in a wide enough range of different environments to be useful, and to absorb from the environment the necessary nutrients to remain functional. Indeed, membranes which are composed purely of amphiphiles are only permeable to small molecules such as sugars, and the charged headgroups of the phospholipids may prevent charged molecules from crossing the membrane. In biological membranes, transport proteins are of great importance, essentially acting as the interface between the intra-cellular and extracellular environments. The selective permeability afforded by these proteins enables the nutrients of cell metabolism to pass into the cell, even against a concentration gradient, and waste to pass out of the cell without leaving the intra-cellular volume vulnerable to contamination [163].

A large number of pore proteins are found in prokaryotic and eukaryotic cells, allowing transport of a diverse collection of molecular species. One pore which has been shown to integrate with vesicle membranes *in-vitro* is the α-hemolysin pore, which is an exotoxin secreted by streptococcus bacteria. This pore integrates into the membranes of red blood cells, causing the lysis of the membrane. The gene for this exotoxin was placed within a vesicle containing a cell free extract including polymerase and ribosome enzymes in [198]. The heptamer proteins were expressed within the vesicle and embedded in the vesicle membrane, porating it and enabling external stores of nucleotides and amino acids to diffuse into the vesicle core. By porating the vesicle in this way the

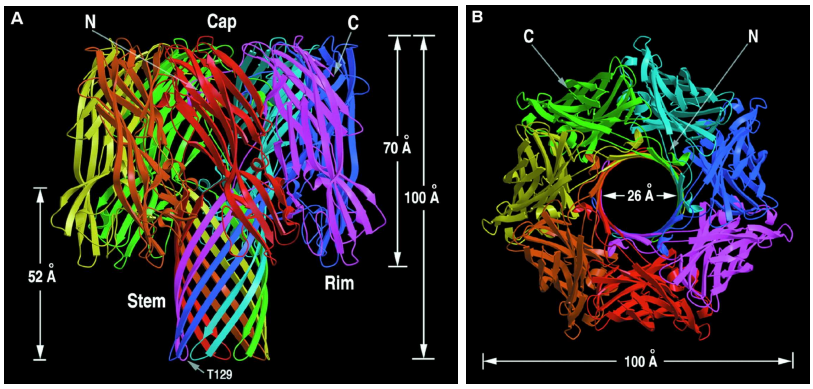


Figure 6.1: The α-hemolysin pore, from [247]. Reprinted with permission from AAAS.

authors were able to extend the duration of gene expression within the vesicle from a few hours to several days. Figure 6.1 shows the structure of the α-hemolysin pore complex.

More complex membrane pore proteins provide the ability to move specific molecular species into and out of the membrane and can be activated by factors internal or external to the cell such as voltage, pH and cell signalling. Expressing a variety of these membrane proteins within the vesicle computing elements will likely be necessary to ensure that the required chemicals for the vesicle computation are available.

Clearly, the selection and expression of pore proteins and their integration into the vesicle membrane will be a very important consideration in the quest to create a system which is autopoietic and can control the input and output of resources involved in metabolism thereby enabling the adaption of the vesicle to new environments. The inclusion in the membrane of the interface for sensor and signalling systems will also be important, and it likely that ligands and other proteins which can influence the expression of genes inside the vesicle will be an integral part of a vesicle computer design. The ability to effectively model and simulate the behaviour of membrane proteins will be a required feature of the vesicle computing simulation and modelling framework, and so the simulation and modelling of pore proteins embedded within vesicle membranes, using the framework is now considered.

The investigation in this chapter involves the simulation of a system of porated vesicles using the DPD and stochastic P systems methods, in which vesicles are self-assembled in the presence of hollow channels which are based on the α-hemolysin heptamer, which integrate into the membrane enabling the unrestricted flow of solvent into and out of the vesicle.

6.2 Simulating Porated Vesicles in DPD

Despite the usefulness of the P systems formalism as an abstract computational device [211], it is not clear if the interactions involving membranes or the membrane structures themselves are models of entities that are feasible *in vitro*. In P systems, regions have no spatial element, and complex

processes such as membrane fission and fusion are conceptualised as atomic operations, rather than complex processes in their own right. Because of this reduction in the level of detail (which is necessary to produce a concise computational formalism) it is desirable to know whether models created in P systems maintain a connection with biological or chemical cells, or if the models are too abstract to effectively analyse models of biochemical systems.

To illustrate the mapping between the two simulation methods, and to show how the two methods can be used together for multi-scale simulation, models of porated vesicles were self-assembled in DPD, and their diffusion properties were then characterised. Models of the same vesicle structure were then created and simulated as stochastic P systems, by parameterising the influx and efflux of solvent in the P systems model to reproduce the results from the DPD model. By expressing the membrane structures as a set of pre-assembled vesicles within molecular simulation, it will be possible to determine numerically if the structural and stoichiometrical dynamics as described in the formalism are stable and represent the original design intention, and gain an understanding into some of the issues involved in implementing the system in vitro. By analysing the dynamic behaviour of the membrane structures in the DPD systems models, it will be possible to alter or constrain P systems to create more realistic vesicle computing models.

Models are created of a very simple diffusive system containing a single vesicle, to study the effect of poration on the rate of diffusion across the vesicle membrane by tracking the movement of solvent particles from within the region to the external environment. Membrane pores vary greatly in complexity and diameter, from passive Porin proteins which select based on molecular size to large, complex adenosine triphosphate (ATP) activated proton pumps and gated ion channels which open or close based on presence or absence of an activator, such as a ligand binding and inducing a conformational change in the protein [163].

6.2.1 Amphiphile Parameters

For all simulated amphiphiles in this work, a model amphiphile is used which is similar to those used in [232] for simulation of fusion of vesicles with a planar bilayer. The amphiphiles are subdivided into volumes of $90\mathring{A}^3$ and the resulting beads are "bonded" using simple Hookean spring forces. The amphiphile configuration is shown in figure 6.2 where the red (darker) particles make up the hydrophilic head, and the green (lighter) particles compose the hydrophobic tails. All bonds between amphiphile particles have a preferred length of 0.7 DPD units[1] and a strength of $100k_bT$. An angle force with strength $6k_bT$ is applied to each sequential triplet of beads in the tail chains, with a preferred angle of 180^o and an angle force with strength $3k_bT$ and preferred angle of 90^o is applied at the point where the two tail chains join the head group. It was found that amphiphiles with these parameters rapidly self-assemble into vesicles.

The conservative force parameters for the different particle types composing the model

[1]The unit length in DPD is the force interaction radius, as a single bead has a volume of $90\mathring{A}^3$, the interaction radius is therefore 6.463Å.

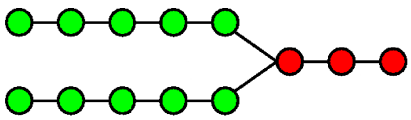


Figure 6.2: The structure of the model amphiphile

amphiphile are given in Table 6.1. The σ and γ parameters are set to 3.00 and 4.5 respectively, resulting in a temperature equivalent to room temperature. The time step, Δt was set to 0.05 DPD time units and the empirical parameter for the Groot Warren integration scheme λ was set to 0.65. To produce a corresponding representation of the P system within DPD, it is necessary to create an initial condition for the simulation containing the required structure of vesicles.

Type	Water	Tail	Head
Water	25	80	15
Tail	80	25	80
Head	15	80	25

Table 6.1: the conservative force interaction parameters for the two-tailed model amphiphile

6.2.2 Modelling Pores

To facilitate the exchange of molecules between compartments we designed pores which assemble into the vesicle membrane. The model pore is a roughly circular inclusion and is constructed from two layers. The outer layer reproduces the hydrophobic/hydrophilic profile of the bilayer, and the inner layer is simply composed of tethered solvent particles, which act as a buffer between the hydrophobic beads in the outer layer and the solvent beads passing through the pore. Without this buffer, it was found that amphiphiles would embed within the pore during the self-assembly process, effectively blocking it as water particles were unable to pass the hydrophobic tails of the amphiphile.

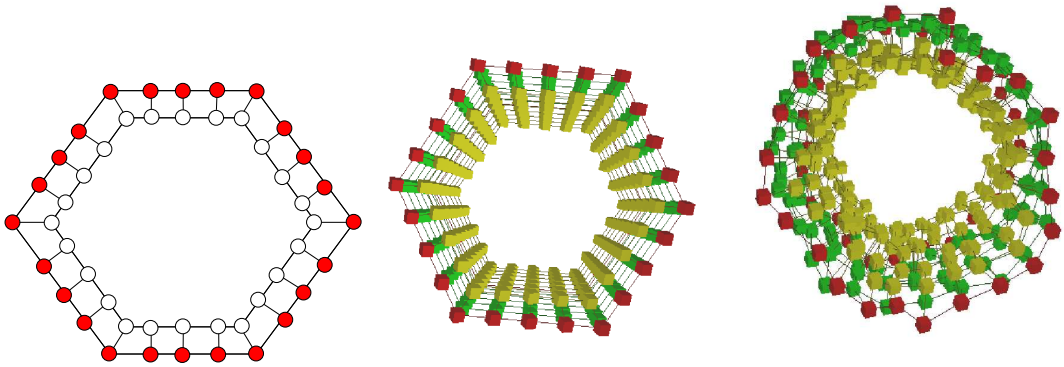


Figure 6.3: A schematic representation of a pore (left), the initial configuration (centre) and the configuration after 1000 time steps (right).

Figure 6.3 shows the pore design, the initial configuration of the pore, and the pore after simulation for 10,000 time steps; the darker particles in the external layer of the pore mimic the properties of the bilayer. All bond lengths between particles having a preferred length of 0.5 DPD units, and the maximum bond strength parameter k for all pore bonds was set to $100k_bT$. Although in the initial configuration the pores are hexagonal in shape, they quickly relax into a circular structure with a diameter of roughly 18Å. Each pore has seven layers. The top and bottom layers contain hydrophilic particles in the outer layer, and the middle five layers contain hydrophobic particles. This profile was chosen to mimic the cross sectional dimensions of the bilayer.

6.2.3 Self-Assembling Porated Vesicles

In order to create the porated vesicle representing the P system region, DPD simulations were performed using the parallel cluster implementation of the code. The initial state of the simulation contained amphiphiles and pore polymers distributed randomly about a cubic simulation space of 50^3 in DPD units with a number density of 3, resulting in simulations containing $375,000$ particles in total. The simulation space was initialised by randomly distributing DMPC amphiphiles, such that 19% of the total number of particles belongs to DMPC polymers. The percentage of particles belonging to pores was then varied in the range of $0.25 - 1.5\%$ in increments of 0.25%. Each simulation was integrated for 200,000 time-steps and vesicles were found to typically form after 100,000 steps. As the number of pores placed initially in the simulation space, the number of pores which integrated into the vesicle membrane also increased. The resulting porated vesicles were then extracted from the simulation output using the *dpddisplay* tool and were used to create the initial conditions for the diffusion simulations. Table 6.2 provides details regarding the vesicles used for the diffusion simulations including estimated radii and the number of pores included in the membrane, and Figure 6.4 shows an image of a vesicle containing 7 pore inclusions, alongside a schematic representation of a P systems model representing that vesicle.

Pores (Total)	Pores (membrane)	Vesicle Diameter (nm)
2	1	15.41
5	2	10.94
8	4	12.40
11	7	15.61
13	11	16.05
16	14	18.10

Table 6.2: Simulations of vesicle formation, with membrane pore inclusions

In order to measure the rate of diffusion, each extracted porated vesicle was placed in a new simulation volume using the *dpdtimestep* tool, and surrounded by solvent such that the correct number of particles was contained within the volume. The internal solvent particles of the vesicle were tagged *yellow* and the vesicle was surrounded by un-tagged *blue* solvent. The interaction

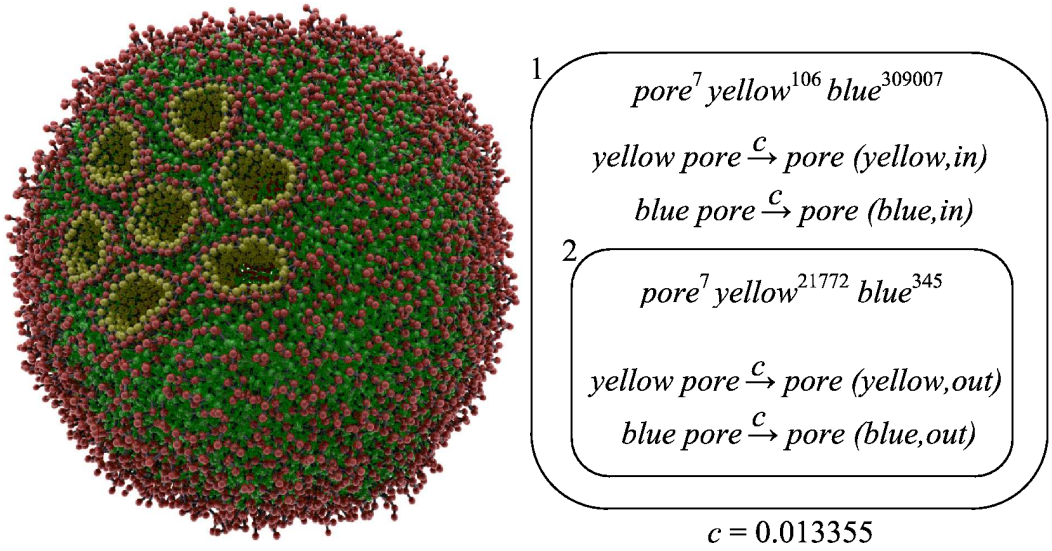


Figure 6.4: The left image shows the result of vesicle self-assembly in the presence of membrane pore inclusions which are included in the membrane as the vesicle forms. The right image shows an example P system representing the DPD porated membrane experiments. The P system contains two regions, the outer one containing the particles in the surrounding environment, and the inner one representing the vesicle. The composition of the multiset of each region represents the initial state of a porated vesicle experiment in DPD, where the inner core of the vesicle contains mostly tagged yellow solvent, and the environment contains mostly untagged blue solvent. The rules represent the transition of the solvent across the vesicle membrane in both directions.

parameters for the yellow and blue solvent particles were the same, and so the distinction was made purely to enable the DPD toolkit software to track the movement of the yellow particles from within the vesicle into the surrounding region. The vesicles were then simulated for 400,000 - 2,400,000 timesteps, Table 6.3 shows number of yellow and blue particles within each vesicle at the start of the simulation, and the length of each simulation in time steps.

Pores (membrane)	Yellow particles	Blue particles	Simulation length (time-steps)
1	21120	173	2,400,000
2	7378	250	600,000
4	10900	169	400,000
7	21772	245	400,000
11	23265	767	400,000
14	32352	2182	400,000

Table 6.3: Initial particles counts within each vesicle

For the vesicles containing only one or two pores, it was necessary to extend the length of the simulation to ensure that the process of diffusion reached an equilibrium. However, for the vesicle which contained only a single pore the rate of diffusion across the membrane was so slow that it was not possible to reach an equilibrium within feasible timescales, and so the simulation is

stopped after 2,400,000 timesteps. A 400,000 timestep simulation took roughly 8 hours to run using 8 nodes on the university of Nottingham cluster in parallel.

In each simulation, a snapshot of the particle and amphiphile positions was taken every 1000 steps, and the *dpdanalysis* tool was used to automatically identify the vesicle in each snapshot and calculate the number of yellow particles contained in the inner vesicle region and the surrounding volume and the results are considered in the next section, where a P system model of the same system is constructed.

6.3 A Stochastic P System Model of Porated Vesicles

As the DPD experiments are quite costly in terms of required computational effort, an attempt was made to use the information derived from the DPD simulation regarding the rate of diffusion of the solvent from the inner core of the vesicle to parameterise a less detailed P systems model which would allow more rapid investigation of the relationship between the pores and the diffusion rate. Conceptually, it is not possible to consider explicitly in P systems any aspect of the membrane composition, or the structure of the pores. However, as in this instance we are only interested in the relationship between the number of pores and the diffusion rate, it is only necessary to consider the effect on the rate of diffusion each pore has, rather than how it has that effect. By using both DPD and stochastic P systems as a multi-scale modelling approach, it is possible to benefit from the best aspects of both approaches, as realistic parameters can be derived from the DPD results, which will then parameterise the computationally less intensive P systems, allowing a much more thorough analysis to be performed within a reasonable amount of time.

An example of the kind of stochastic P systems model of membrane diffusion which was used for the P systems simulations in this chapter is shown below:

$$
\begin{aligned}
&\mathcal{SP}_{pv} = (M_{pv}, \mu_{pv}, L_{pv}, I_b, R_b) \quad \text{where} \\
&M_{pv} = \{yellow, blue, pore\} \\
&\mu_{pv} = [\ []\ ] \\
&L_{pv} = \{surrounding, vesicle\} \\
&I_{surrounding} = \{blue^{309831}, yellow^{395}\} \\
&I_{vesicle} = \{blue^{191}, yellow^{21123}\} \\
&R_{surrounding} = \\
&\left\{ \begin{array}{l} [yellow, pore]_{surrounding} \xrightarrow{0.001} [yellow]_{vesicle}\,[pore]_{surrounding} \\ [blue, pore]_{surrounding} \xrightarrow{0.001} [blue]_{vesicle}\,[pore]_{surrounding} \end{array} \right\} \\
&R_{vesicle} = \\
&\left\{ \begin{array}{l} [yellow, pore]_{vesicle} \xrightarrow{0.001} [yellow]_{surrounding}\,[pore]_{vesicle} \\ [blue, pore]_{vesicle} \xrightarrow{0.001} [blue]_{surrounding}\,[pore]_{vesicle} \end{array} \right\}
\end{aligned}
$$

Region *surrounding* represents the fluid surrounding the vesicle, and region *vesicle* repre-

sents the vesicle and its encapsulated solvent. For the surrounding region, the rules state that an interaction occurs between a blue or yellow solvent particle and a pore, and the solvent particle is transmitted into the vesicle's region, with the pore remaining in the surrounding region. Likewise in the vesicle, the yellow or blue solvent particles interact with the pores and travel out of the vesicle into the surrounding region. Initially, the surrounding regions contains only a minimal amount of yellow solvent particles, just as in the DPD simulation, and the encapsulated volume of the vesicle contains a majority of yellow particles. In this instance the model is representing a membrane with 7 pores embedded in it, so the number of pore objects in each region is 7. Initially, the stochastic rate constant c is the same for all of the rules in the system and was set to an arbitrary value of 0.001. This value ensured that the propensity of each rule was dependent solely on the number of yellow particles, blue particles and pores within each membrane.

The P systems and DPD models were simulated for an equivalent amount of time. Figure 6.5 shows the movement of solvent objects between regions in both simulations. In both cases, yellow particles initially moved out of the vesicle and blue particles enter the vesicle due to high concentrations of these particles in regions 2 and 1 respectively and both simulations reached equilibrium after approximately 1,500,000 time steps. However, there was a clear difference between the number of yellow and blue particles in each region in the results of the DPD and P systems simulations. At the end of the P systems simulation, the number of yellow particles in both regions had reached an equilibrium where the number of yellow particles was roughly equal in the vesicle core and the surrounding volume, with the number of yellow particle in each region being roughly half of the total (about 11000 particles). In contrast the results from the DPD simulation showed that once the system had reached equilibrium, the vesicle region contained far fewer yellow particles than the surrounding region, with the former containing roughly 1500 yellow particles and the latter containing around 22000.

This disparity can be explained by considering the meaning of the stochastic rate constant c, which was set to be the same for all reactions in the simulation. The stochastic rate constant is used in combination with the number of possible configurations of reactants, to determine the propensity of the reaction, as explained in Chapter 3. According to Gillespie's explanation of the stochastic simulation algorithm [104] the c parameter represents the probability of a pair of reactants coming into contact and reacting within the next infinitesimal increment of time. Clearly this probability is dependent on the physical properties of reactants, the temperature of the system, and the volume in which the reactants are diffusing. By setting the rate constants for all reactions in the model to the same value, the implicit assumption is made that both regions have the same volume. This is clearly not the case in DPD, as Table 6.3 indicates that the diameter of the vesicle is 15.41nm, giving an estimated volume of $1916nm^3$, whereas the volume of the entire simulated volume is $39304nm^3$. Considering this fact, it should be expected that in the DPD results the concentration, rather the absolute number, of yellow particles inside the vesicle region is similar to the concentration of yellow particles outside the vesicle region once equilibrium is reached. Estimates based on the values stated

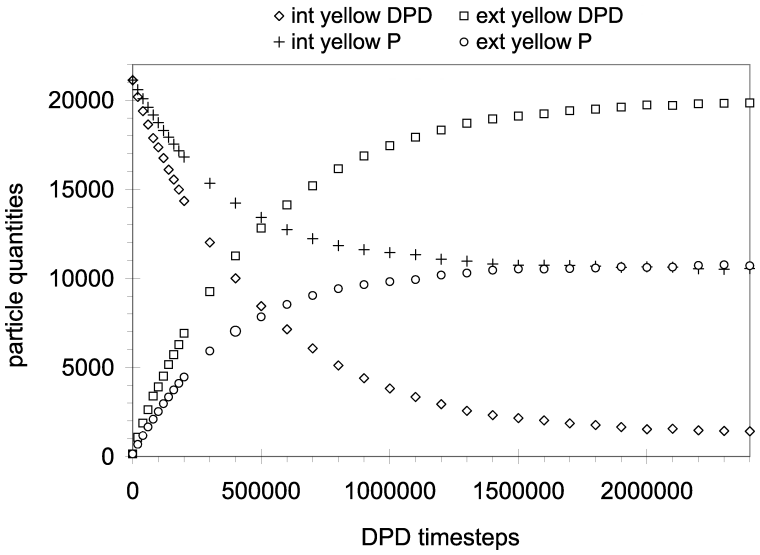


Figure 6.5: Equilibration in the initial P system model does not match DPD.

above indicate this to be the case. The mole fraction of yellow particles in the vesicle is 0.217 and 0.196 in the surrounding region. If it is assumed that the volume of the vesicle is constant throughout the simulation, then it is possible to integrate the volume information into the P system model and determine a more accurate parameterisation.

6.3.1 A Refined P Systems Model

In Chapter 3 it is noted that in the stochastic simulation algorithm, the probability of a collision occurring between a pair of reactants within an infinitesimal time increment is given by Equation 6.1.

$$P_{coll} = \frac{V_{coll}}{V} \tag{6.1}$$

Since the value for the volume of each region V is known it is only necessary to determine a value for the collision volume swept out by the water and pore pair V_{coll}, to parameterise the pore transport rule in each compartment correctly. The value for V_{coll} could be calculated by determining the average relative velocity between the solvent particles and the pore inclusion, and estimating the radius for the pore inclusion based on the average position of the particles, but this would require averaging over a large ensemble of simulations to determine a sufficiently accurate estimate, which would be very computationally expensive in DPD. Instead, a different approach was taken, where a trial and error based estimation of V_{coll} was performed by modifying it to find a value in which the output time series of the stochastic P system simulation was similar enough to the time series from the DPD simulation. The estimation was performed by hand, but could easily be automated using a simple optimisation technique. Using this method, the value of 0.013355 was determined for the collision volume, and a new P systems model of the DPD porated vesicles was created with the

following rate constants for the pore transport rules.

$$c_{vesicle} = 0.013355/7097 = 1.8881 \times 10^{-6} \tag{6.2}$$

$$c_{surrounding} = 0.013355/117903 = 1.1132 \times 10^{-7} \tag{6.3}$$

With these stochastic rate constants for the interaction between the pores and water particles the behaviour of the P system closely matched that of the DPD simulations. Figure 6.6, shows the result of P systems simulations where the rate constants for the rules were set using the above volume of collision and the volumes determined from the DPD vesicles. It can be observed that simulations with more pores had a greater initial rate of yellow efflux. In each case, the final concentration of yellow particles is dependent on the volume of the vesicle (as in the DPD model, all of the encapsulated particles are made yellow initially).

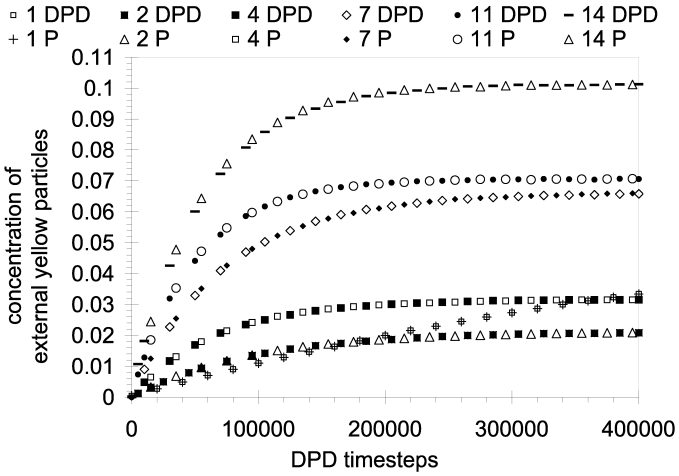


Figure 6.6: A plot of the concentration of external yellow particles from both P system and the DPD simulations over 400,000 time steps which a range of pores.

Figure 6.7 plots of the change in concentration over time during the first 50,000 steps against the number of pores in the simulation. The rate of efflux for yellow particles increases linearly with the number of pores in the membrane, increasing by around 0.0037 with each additional pore. The results show that despite the difference in scale, the P systems model can be calibrated to match the observed behaviour of diffusion in the DPD model. By linking the two in this way, the scale gap between the abstract model and more realistic low level simulation has been bridged. This work also illustrates some of the benefits of constructing vesicle computing models using a multi-scale modelling framework, as the less detailed technique can be used to quickly estimate parameters, and eradicate errors in the assumptions and proposed models in a manner which is less computationally expensive than the DPD method.

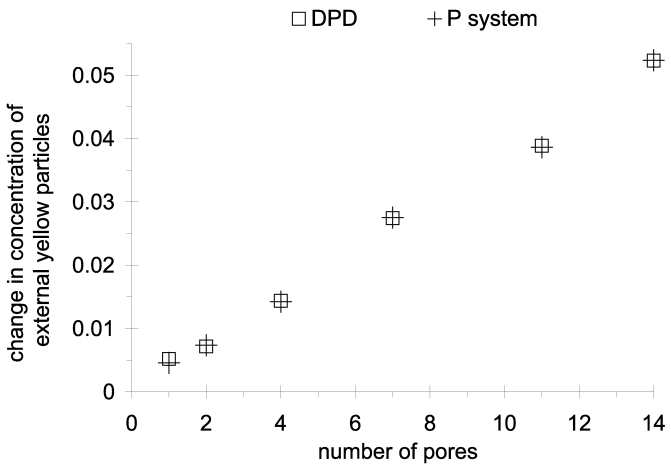


Figure 6.7: The rate of yellow efflux is proportional to the number of pores in a membrane.

6.4 Comparison With Laboratory Experiment

The results from simulation indicated a linear relationship between the number of pores present in the membrane and the rate of diffusion. As a step towards a physical implementation of the liposome logic approach, vesicles were prepared which encapsulated a fluorescent reporter by Francisco Fernádez-Trillo at the University of Nottingham School of Pharmacy. Prior literature has shown that several amphiphilic moieties can be employed for encapsulation of nucleic acids and proteins, some of which have proven useful as bioreactors, drug delivery vehicles or as early-stage protocells [198, 183, 224, 256]. For instance, some of the amphiphiles employed for the encapsulation of DNA show interesting properties such as the ability to respond to an external stimulus, such as pH [164, 138]. For the vesicle experiments presented here polyethylene glycol-polylactide (PEG-PLA) vesicles developed by Discher et al. [138] were used. Vesicles formed from the PEG-PLA di-block copolymer amphiphiles have an interesting property, in that the rate at which molecules encapsulated within the vesicle are released is a function of the pH of the surrounding medium, as a decrease in pH results in the formation of pores in the vesicle membrane as a result of the hydrolysis of the polyester block [11]. An EO_{50}-b-LA_{50} copolymer was synthesized, and large multilamellar vesicles of approximately 125 nm in diameter were formed (Figure 6.8).

Large EO_{50}-b-LA_{50} unilammelar vesicles loaded with 5(6)-Carboxyfluorescein as a fluorescent reporter were prepared and evaluation of the fluorescence as a function of time showed that the vesicle ensapsulent was slowly released, indicating a measurable response to an external pH stimulus. The results of these experiments are shown in Figure 6.9 and can be compared with those from Figure 6.6, in which the diffusion rate of encapsulated particles within vesicles containing different numbers of membrane pores was determined in simulation. The relative intensity of the fluorescent reporter can be seen to be qualitatively similar to change in concentration of encapsulated particle

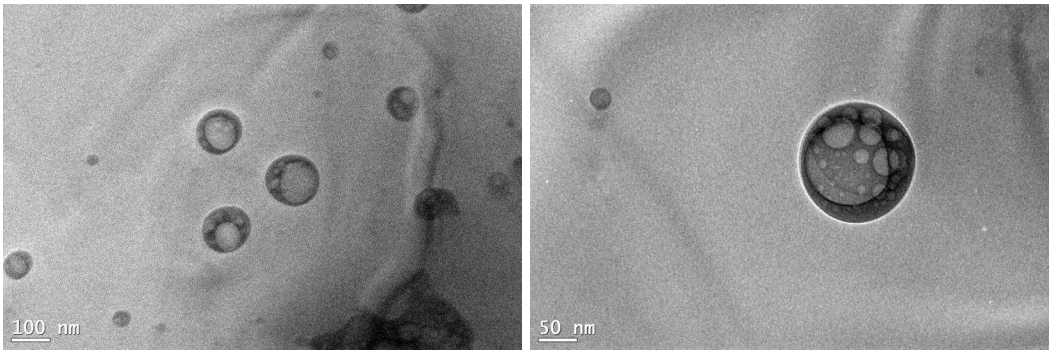


Figure 6.8: Cryogenic Transmission Electron Microscope Images of EO_{50}-b-LA_{50} LMVs

within the simulated vesicle, although occurring over a much longer time-scale.

6.5 Summary

In this chapter the vesicle computing simulation framework has been used to investigate a possible mechanism for the transmission of information between vesicle computing entities, the diffusion of chemical signalling molecules.

By coupling the P systems formalism with a coarse grained particle dynamics simulation, and calibrating the high level model with results from the low level simulation I have shown that designs can be expressed quickly as P systems, and high level stochastic analysis performed. If the design shows promise, then the system can be represented within DPD as a set of vesicles, and the dynamics of the system fully explored. I show that this method is viable by analysing diffusion in a simple system of porated membranes, and by refining the model described we produce a stochastic P system that correlates well with the behaviour from the DPD simulation. This P system can then be used in more complex designs of artificial life systems, or for gaining insight into the behaviour of the system over longer time scales.

Although the system being modelled is very simplistic when compared to the highly functionalised protein pore inclusions found in biological cell membranes, the results illustrate a proof of concept for the modelling framework.

When vesicle computers are initially constructed, the understanding of the processes involved in porated diffusion will be important, as it is likely that the vesicle computers will be initially constructed to express proteins α-hemolysin, which will porate the vesicle membrane and enable a simple diffusive flow of nutrients to the vesicle metabolism, as illustrated by encapsulated gene expression work of [198]. As the complexity of vesicle computer designs increases, more functional pore proteins could be expressed in the vesicle membrane, such as those performing active transport. At present the behaviour of such pores cannot be explored using the DPD method, but further extensions could be made to the method to add this functionality, and the effect of these more complex

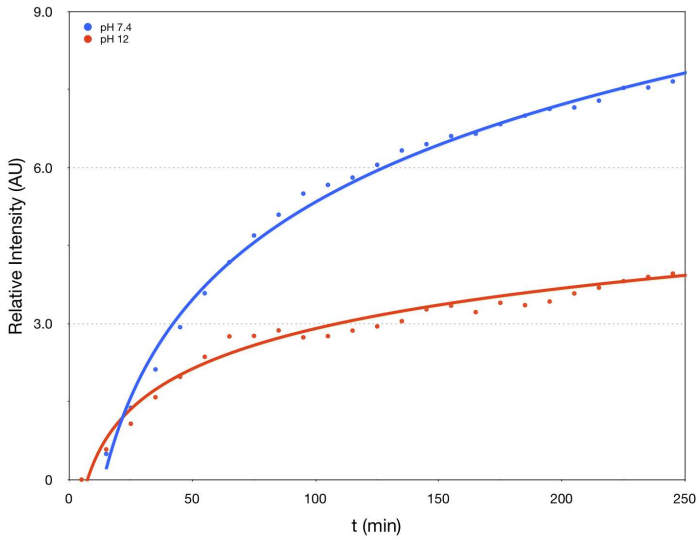


Figure 6.9: The graph shows the release of CF from $EO_{50} - b - LA_{50} - LUVS$ as a function of time, measured at two different pH values. The increase in acidity results in a greater number of pores in the membrane which increases the rate at which the fluorescent reporter diffuses out of the vesicle. The results are qualitatively similar to those from simulation in figure 6.6, which showed the change in concentration of encapsulated particles over time, as they diffused out of the vesicle.

pores could be investigated using a stochastic P systems model. This study shows the efficacy of the simulation and modelling framework at exploring simple pore based membrane diffusion processes.

In the next chapter, the application of the vesicle computing simulation framework to the computational aspects of vesicle computing is presented in the form of simulations of simple chemical reaction based logic gated, encapsulated within vesicles. These simple logic gates were then used as modules, in a similar fashion to the biobricks approach in synthetic biology, to construct more complex computational devices, including a flip-flop.

Chapter 7

Liposome Logic

In this chapter models of vesicle computation are specified and simulated using the vesicle computing simulation and modelling framework. Chemical reactions, mimicking the mechanics of genes expression and regulation in prokaryotes are specified to produce simple logic gate functionality. These reactions are then encapsulated within vesicles in DPD simulations, and the long term dynamics of vesicle computation are studied using SSA and probabilistic model checking. The results of these simulations support the hypothesis that such reactions could be encapsulated within phospholipid vesicles to perform useful computation in vitro.

7.1 Introduction

In living cells, complex behaviour is achieved by an exquisite tuning of biological regulatory networks (BRN), such as gene expression, signalling and metabolic networks. Certain BRN motifs [14], patterns of gene regulation which occur far more frequently within natural cell genomes than in randomly generated regulatory networks, have been identified and these motifs produce functionality such as oscillation, feed forward and feed back loops and simple Boolean logic. The combination of these motifs can produce complex decision making behaviours in cells, such as chemotaxis towards an external signal molecule (e.g. glucose in E. *coli* bacteria), quorum sensing, etc.

In this chapter, the vesicle computing simulation framework is used to extend the study of the membrane and communication dynamics to consider the computational reactions which are encapsulated within the vesicle. The approach is similar in some respects to membrane computing [211], in which computation can take place by applying reaction rules to chemicals within encapsulated volumes. In contrast with the most common "top-down" approach to synthetic biology, a "bottom-up" approach is taken in which the synthetic cell is built with engineered components of manageable complexity. The aim is to avoid many of the difficulties that arise when attempting to modify a highly complex system (e.g. a living cell). To avoid this complexity trap, the computational device is built from scratch utilising any biochemical or chemical system that may be relevant in solving a computational problem. In other words the aim is to chart a route for a vesicles-based

computer that jettisons the billion year evolutionary baggage of bacteria and instead, mimics the key critical components that could make a vesicle computer possible: a simple membrane and a simple regulatory-like circuit. That is, the "legacy system" present in current top-down synthetic biology projects is discarded in favour of an approach in which the complexity of every single component is known (and characterised) in advance.

A proof of concept model of simple computational entities, that could combine the beneficial features of both the DNA and cell computing approaches is proposed. In this computing paradigm, individual cells perform a small part of a computation in a highly asynchronous fashion with communication taking place only between cells which are within a short distance from one another (so called amorphous computation, [3]). In particular, cellular logic NOT and AND gates were first characterised in detail by Ron Weiss et al. [262] in-vitro and with "bioSPICE", an ODE based modelling technique. Since then, several small scale systems have been constructed, either in the lab or in simulation [16]. Other examples of in-vitro implementations of cellular computing systems include band detectors, coupled oscillations [28, 29] and, more recently, a solution to the three vertices Hamiltonian path problem [31]. The approach retains all the advantages of amorphous computing at the nanoscale (e.g., redundancy, massive parallelism, asynchronous local processes, self-organisation, etc.) but by starting from the bottom up with engineered components of prespecified and limited complexity, one avoids unnecessary biological nuisances from the start. Moreover, by turning to chemical cellular-like constructs, compartmentalisation and orthogonality are maximised while crosstalk minimised. Thus the proposed approach might provide a route to, e.g., more reliable programmable chemical communication and drug delivery systems [209, 99].

Vesicles can also be used for encapsulation and implementation hiding when constructing a vesicle computer, as hierarchical (i.e. nested) structures of membranes could be created with clearly defined inputs and outputs, that create boundaries around functionality just as organelles contain specific functions in eukaryotic cells. This kind of compartmentalisation will enable interference between BRNs to be minimised, and the need for multiple promoter sequences/transcription factors to be reduced. The scalability of the techniques presented was analysed as the complexity was increased from relatively simple NOT and NAND gates to SR-Latches, D Flip-Flops all the way to 3 bit ripple counters.

This chapter presents a rigorous computational simulation as a proof of concept for the vesicle computing approach, performed with the vesicle computing simulation framework. The vesicle computing model is specified and simulated in a full 3D particle simulation using Dissipative Particle Dynamics (DPD). First, three logic gates, AND, OR and NOT, based on the transcriptional logic reported by Silva-Rocha et al. [239] were implemented within liposomes and simulated with DPD. The behaviour of the logic gates are compared with control experiments in which the reactions and particle types are not encapsulated within a liposome to investigate how encapsulation effects the dynamics of the reactions. Second, more realistic logic gates designs are created based on parameters for gene regulation in the literature, and both DPD and Stochastic P systems are used to simulate

larger systems composed of multiple logic gates, such as oscillators, and memory units. The models are designed to be modular, so that they can be easily duplicated and connected together to create new, more complicated models.

By creating *modular* logic gates that behave in well characterised ways, it might be possible to abstract away some of the biological detail when designing more complex synthetic biology systems [18]. In doing so, the behaviour of a composite system becomes more predictable and designs can be constructed and prototyped in-silico before attempting to implement them in the lab. Just as an electrical engineer can construct circuits from modules with common inputs and outputs without consideration of internal module construction, the standardisation of biological components proposed by T.F. Knight, D. Endy, R. Weiss and others [81, 139, 124, 228], and exemplified in the MIT biobricks project [230], may allow a bioarchitect to construct biological systems with prespecified phenotypes in a more scalable way. Figure 7.1 shows the different logic components that were simulated and the simulation technique used to perform the simulation.

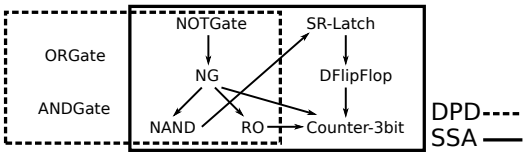


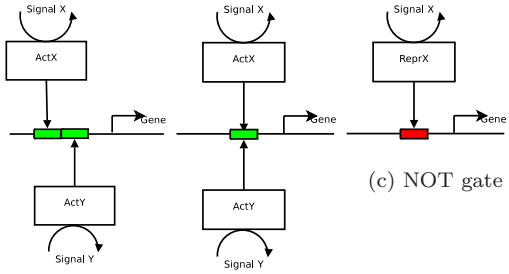
Figure 7.1: The diagram indicates which modules were simulated using which simulation technique (e.g. DPD or SSA), and the relationship between the different models simulated using the pipeline, an arrow pointing from one module to another indicates that the module at the arrow's source is used by the other module.

7.2 Gene Regulatory Network Logic Gates in DPD

For the first investigation of model logic gates encapsulated within vesicles, models of three different types of logic gates were created, an AND gate, an OR gate and a NOT gate. The models were intended to be general and qualitative, similar in style to those used by Magnasco to show the Turing universality of chemical reactions [171]. The logic gates were based on the behaviour of biological protein expression in prokaryotes, but without using rates measured from any specific biological system. Thus, in principle at least, they could be implemented through a bottom-up synthetic biology route.

The logic gates which were used as a case study for implementation within the simulated liposomes are based on the transcriptional regulation of genes that occurs in prokaryotic cells. For example, the expression of a gene starts with the binding of an RNA polymerase enzyme to the promoter of the gene sequence. The gene is then transcribed into messenger RNA which is translated by a ribosome into a protein. The expression of the gene can be regulated by the presence or absence of a repressor which binds to the promoter. If the repressor is bound to the promoter then the polymerase cannot transcribe the gene sequence. The regulation of gene expression in the models

is driven by the presence or absence of auto-inducer signalling molecules, which react with the transcription factor proteins resulting in the activation or deactivation of the protein. In chemical terms, the interaction between the autoinducer and the transcription factor produces a change in the protein's conformation, either enabling or preventing it from binding to the gene sequence and regulating gene expression. Figure 7.2 shows the pathways of interaction for the AND, OR and NOT gate. In this simplified qualitative model of gene expression, the mechanism by which the transcription and translation occurs is not included, and is replaced with a single reaction. The degradation of the expressed proteins in the system is also ignored.



(a) AND gate (b) OR gate (c) NOT gate

Figure 7.2: The gene interactions representing the three logic gates: rectangles represent activators/repressors. Arrows show the binding of the activator or repressor to the gene promoter (green for activation, red for repression). Curved arrows show the enabling of the activator/repressor by the given signal molecule.

The rates for the DPD reactions representing the logic gates were not based on any particular observation or parameter set, but were instead chosen to ensure that the reactions would occur within the time scale of the simulation. DPD simulations typically do not simulate more than a second of real time, clearly far less than the amount of time for the transcription and translation reactions to occur in biology, where the time taken to express a protein ranges from minutes to hours. However, the work in this chapter is intended as a proof of principle for the inclusion of logic gates within vesicles, and that doing so has a beneficial effect on the dynamics of such systems. Thus in vivo time scales do not translate to the system. That is, as a biological or non biological implementation approach for liposome logic is considered it is the *structure* of the gene circuits for the AND, NOT and OR gates rather than a cell biology specific model that is important. Moreover, words such as "gene" and "gene circuits" are used in the sense of (1) an object that encodes information, but that it does necessarily not have to be a nucleic acid based information store, (2) converts that information into an output, but again for a vesicle computer this does not have to be the familiar RNA $\rightarrow$ protein route, etc. The logical operation is the conversion of the information into an output requiring an energy input and yielding an output. With this in mind, "fast" and "slow" reactions are defined with the reaction constant ω set to 0.1 and 0.01 respectively. Reactions between signal molecules and activators/repressors, the binding of activators/repressors to genes and

the production of the output protein were "fast" and the decomplexation of activators/repressors from genes was "slow".

The model AND gate is based on a gene that requires two activator proteins to bind to the promoter region before transcription can occur. The activator proteins are initially inactive, but are activated by two different signal molecules, X and Y respectively and once enabled are able to bond to the gene. Transcription can begin only when both activators are bound to the gene. The AND gate is modelled in DPD as the following module:

$$
\begin{aligned}
&And(\{sigX, sigY, actX_i, actY_i, actX, actY, gene, prot\}, \{c_1, c_2\}, \{b\}) = \\
&\quad [\ sigX + actX_i\ ]_b \xrightarrow{c_1} [\ actX + solvent\ ]_b \\
&\quad [\ sigY + actY_i\ ]_b \xrightarrow{c_1} [\ actY + solvent\ ]_b \\
&\quad [\ gene + actX\ ]_b \xrightarrow{c_1} [\ gene.actX + solvent\ ]_b \\
&\quad [\ gene + actY\ ]_b \xrightarrow{c_1} [\ gene.actY + solvent\ ]_b \\
&\quad [\ gene.actX + actY\ ]_b \xrightarrow{c_1} [\ gene.actX.actY + solvent\ ]_b \\
&\quad [\ gene.actY + actX\ ]_b \xrightarrow{c_1} [\ gene.actX.actY + solvent\ ]_b \\
&\quad [\ gene.actX + solvent\ ]_b \xrightarrow{c_2} [\ gene + actX_i\ ]_b \\
&\quad [\ gene.actY + solvent\ ]_b \xrightarrow{c_2} [\ gene + actY_i\ ]_b \\
&\quad [\ gene.actX.actY + solvent\ ]_b \xrightarrow{c_2} [\ gene.actX + actY_i\ ]_b \\
&\quad [\ gene.actX.actY + solvent\ ]_b \xrightarrow{c_2} [\ gene.actY + actX_i\ ]_b \\
&\quad [\ gene.actX.actY + solvent\ ]_b \xrightarrow{c_1} [\ gene.actY + prot\ ]_b
\end{aligned}
$$

The model NOT gate is based on the deactivation of a repressor protein by a signal molecule. If the repressor is active and is able to bind to the gene, no transcription can occur. The presence of the X signal molecule changes the conformation of the repressor protein such that it can no longer bind to the gene, allowing transcription to occur. The NOT gate is modelled in DPD with the following stochastic P system module:

$$
\begin{aligned}
&Not(\{sigX, repX_i, repX, gene, solvent, protein\}, \{c_1, c_2, c_3, c_4\}, \{b\}) = \\
&\quad [\ sigX + repX_i\ ]_b \xrightarrow{c_1} [\ repX\ ]_b \\
&\quad [\ gene + repX\ ]_b \xrightarrow{c_1} [\ gene.repX\ ]_b \\
&\quad [\ gene.repX\ ]_b \xrightarrow{c_2} [\ gene + repX_i\ ]_b \\
&\quad [\ gene\ ]_b \xrightarrow{c_1} [\ gene + protein\ ]_b
\end{aligned}
$$

The model OR gate is the same as the model AND gate, except that transcription can occur when

either one of the two activators are bound to the gene. The OR gate is modelled in DPD with the following reactions:

$$
\begin{aligned}
&Or(\{sigX, sigY, actX_i, actY_i, actX, actY, gene, prot\}, \{c_1, c_2\}, \{b\}) = \\
&\quad [\ sigX + actX_i\ ]_b \xrightarrow{c_1} [\ actX + solvent\ ]_b \\
&\quad [\ sigY + actY_i\ ]_b \xrightarrow{c_1} [\ actY + solvent\ ]_b \\
&\quad [\ gene + actX\ ]_b \xrightarrow{c_1} [\ gene.actX + solvent\ ]_b \\
&\quad [\ gene + actY\ ]_b \xrightarrow{c_1} [\ gene.actY + solvent\ ]_b \\
&\quad [\ gene.actX + solvent\ ]_b \xrightarrow{c_2} [\ gene + actX_i\ ]_b \\
&\quad [\ gene.actY + solvent\ ]_b \xrightarrow{c_2} [\ gene + actY_i\ ]_b \\
&\quad [\ gene.actX + solvent\ ]_b \xrightarrow{c_1} [\ gene.actX + prot\ ]_b \\
&\quad [\ gene.actY + solvent\ ]_b \xrightarrow{c_1} [\ gene.actY + prot\ ]_b
\end{aligned}
$$

The models are fully specified as stochastic P systems which use the AND, OR and NOT gate models, shown in figure 7.3. The stochastic P system specifications are then translated into initial states for a DPD simulation.

7.2.1 Creating the Liposomes and Simulation of Initial Conditions

The transcriptional logic gate reactions were simulated in two situations, firstly with all particles moving freely in bulk solution (control experiment), and secondly with the gene and activator particles encapsulated within the inner volume of the liposome. The liposomes were created by randomly placing the model DMPC phospholipid polymers described in Chapter 5 into a volume of $50r_c^3$ with a number density $\rho = 3.0$ such that 18% of the total number of particles (375000) in the simulation volume composed the polymers, creating 5192 polymers in total. The system was then evolved for 10000 DPD time units (with a time step length of $dt = 0.05$), and liposomes were found to have self-assembled at around $\tau = 6250$. The vesicle formation simulation took roughly 2 hours using the CUDA implementation of the DPD software running on an Nvidia 8800 GTX graphics card.

The initial states for the logic gate experiments were created by extracting the self-assembled vesicles using the *timestepanalyser* tool, and then loading the particles and polymers of a pre-computed liposome into a new simulation space, modifying the types of solvent particles encapsulated within the inner volume such that the required gene/promoter types were encapsulated using the *dpdtimestep* tool. The inner volume of the liposome contained 13884 solvent particles. Table 7.1 shows how many particles of each type were placed within the liposome inner volume.

Table 7.2 shows the conservative force parameter matrix for all simulations. The large maximum repulsion value (104) between the particles representing the lipid of the membranes and

$$
\begin{aligned}
\mathcal{SP}_{And} = (M_{And}, &\mu_{And}, L_{And}, I_{bact}, R_{bact}) \quad where \\
M_{And} &= \{sigX, sigY, actX_i, actY_i, actX, actY, gene, prot\} \\
\mu_{And} &= [\ ] \\
L_{And} &= \{vesicle\} \\
I_{vesicle} &= \{gene^5, actX_i^{694}, actY_i^{694}\} \\
R_{vesicle} &= \{And(\{sigX, sigY, actX_i, actY_i, actX, actY, gene, prot\}, \{1.0, 0.01\}, \{vesicle\})\}
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{SP}_{Or} = (M_{Or}, &\mu_{Or}, L_{Or}, I_{bact}, R_{bact}) \quad where \\
M_{Or} &= \{sigX, sigY, actX_i, actY_i, actX, actY, gene, prot\} \\
\mu_{Or} &= [\ ] \\
L_{Or} &= \{vesicle\} \\
I_{vesicle} &= \{gene^5, actX_i^{694}, actY_i^{694}\} \\
R_{vesicle} &= \{Or(\{sigX, sigY, actX_i, actY_i, actX, actY, gene, prot\}, \{1.0, 0.01\}, \{vesicle\})\}
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{SP}_{Not} = (M_{Not}, &\mu_{Not}, L_{Not}, I_{bact}, R_{bact}) \quad where \\
M_{Not} &= \{sigX, reprX_i, reprX, gene, prot\} \\
\mu_{Not} &= [\ ] \\
L_{Not} &= \{vesicle\} \\
I_{vesicle} &= \{gene^5, reprX_i^{694}\} \\
R_{vesicle} &= \{Not(\{sigX, reprX_i, reprX, gene, prot\}, \{1.0, 0.01\}, \{vesicle\})\}
\end{aligned}
$$

Figure 7.3: Stochastic P system definitions for the logic gate models for the qualitative investigation encapsulated logic.

Type	AND Count	OR Count	NOT count
Solvent	12490	12490	13184
activator X (inactive)	694	694	694
activator Y (inactive)	694	694	0
gene	5	5	5

Table 7.1: The count of each particle type placed within the inner volume when initialising the liposome for the logic gates experiments.

the genes and activators ensure that particles of these types are held within the liposome inner volume, as they cannot diffuse across the membrane. An a value of 78.00 gives the types the same immiscibility as water at room temperature.

The initial state was then loaded into the software, and after an equilibration period of 50 time units, in which no reactions were allowed to occur, the system was evolved for 1000 DPD time units, and the number of particles of each type was counted at the end of each time unit, in order

Type	S	T	H	Sig X,Y	Act X,Y	Repr X	Gene X,Y	Output
S	78	104	78	78	78	78	78	78
T	104	78	104	78	104	104	104	78
H	78	104	86.7	78	78	78	78	78
Sig X,Y	78	78	104	78	78	78	78	78
Act X,Y	78	104	78	78	78	78	78	78
Repr X	78	104	78	78	78	78	78	78
Gene X,Y	78	104	78	78	78	78	78	78
Output	78	78	78	78	78	78	78	78

Table 7.2: The conservative force a parameter for the encapsulated AND gate simulations. S, T and H are the solvent, DMPC tail and DMPC head types respectively.

to record the dynamics of the system. To account for the stochastic nature of the dynamics, each simulation was performed three times, and each run took 37 minutes in CUDA. This process was repeated for each set of inputs to the logic gate, in order to observe the dynamics of the system in response to the presence or absence of the different signal inputs.

7.2.2 Logic Gate Dynamics

The following figures show the time series of the number of particles (for each particle type) for each logic gate for both the control and the within-liposome experiments. The graphs in figure 7.4 show that when the X signal molecule is present, the number of output proteins present at the end of the simulation is reduced due to the binding of the signal molecule to the repressor, which prohibits production of the output molecule. However the production of the output protein is not inhibited entirely, as the repressor can decomplex from the gene. The effect of the encapsulation of the NOT gate is to increase the likelihood that the repressor and gene can come into contact and form a complex. Therefore the overall expression rate of the protein is, as expected, lowered.

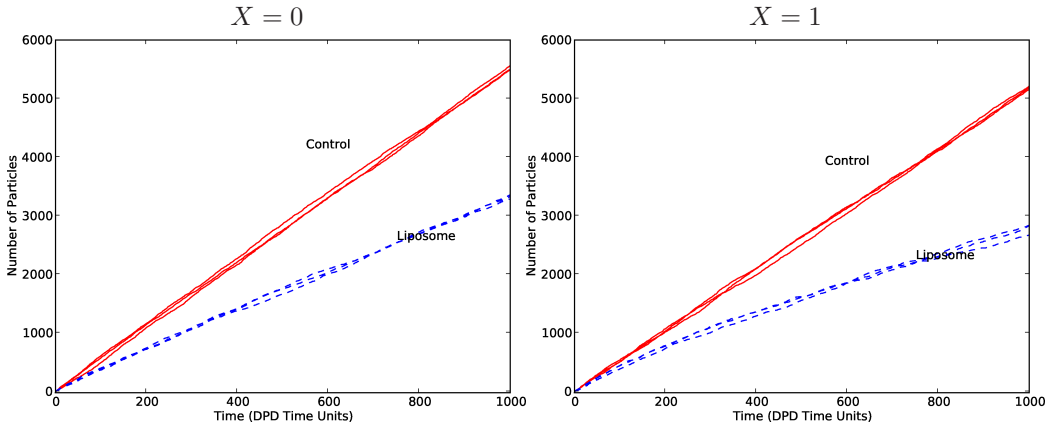


Figure 7.4: NOT Gate Results

Figure 7.5 shows the results for the control and vesicle AND gate experiments respectively. The figures show that the creation of the output gene occurs very infrequently in the control experiments, and in fact only a single output protein is created in all three runs. This is in contrast with the results from the runs where the AND gate is encapsulated within the vesicle, where on average 275 output proteins were produced by the end of each run, due to the maintenance of the region of high activator and gene concentration within the vesicle volume. The AND logic gate, is shown to function correctly, as the output protein is only produced when both input signals are present.

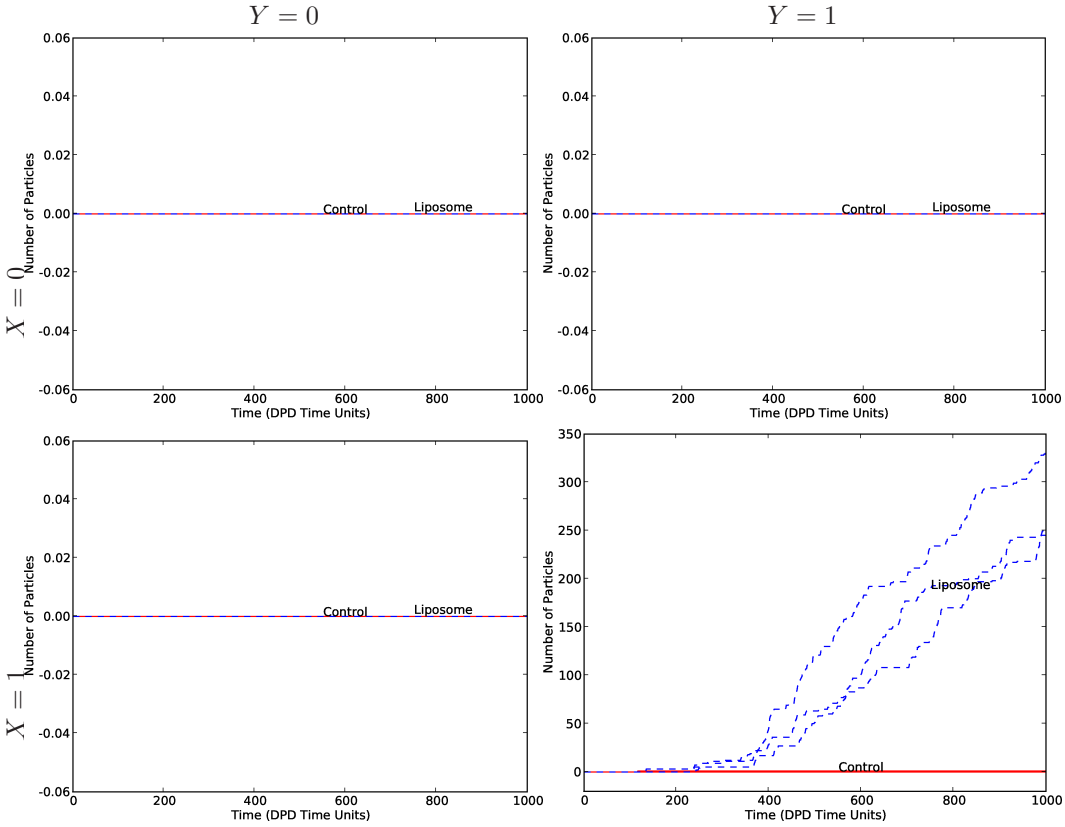


Figure 7.5: AND Gate Results

Figure 7.6 shows that the OR gate reactions occurred at a faster rate than the AND gate reactions, most likely because the pathway from signal to output protein involves only two reactions (as opposed to three in the case of the AND gate) in order to generate proteins. The OR gate reactions functioned correctly in terms of modelling the OR logic gate in the vesicle and control experiment as the output protein was produced when either or both of the inputs was present. In both the control and vesicle experiments, the number of output particles produced was greater when both input signals were present, as both activators are enabled and could react with the gene rather than only one for the single input cases.

The results from the logic gate experiments indicated that encapsulation within the li-

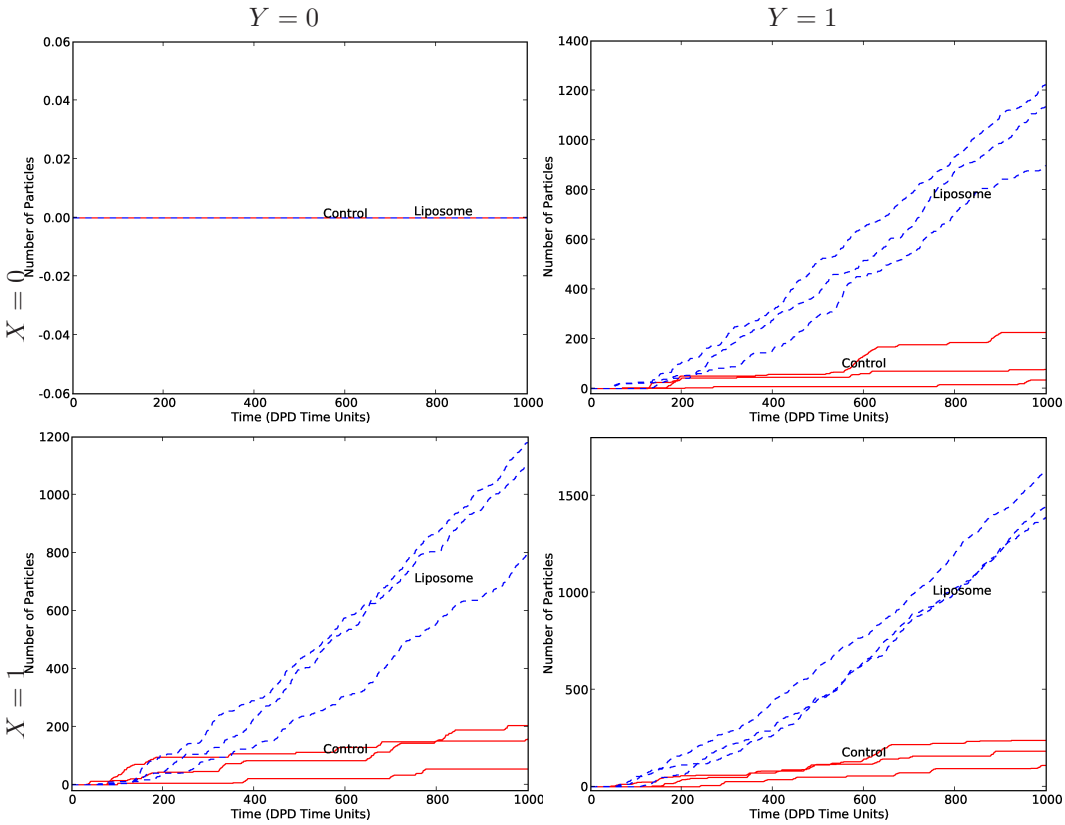


Figure 7.6: OR Gate Results

posome may have had a catalytic effect on the dynamics of the logic gate reactions, due to the encapsulation of the particles which produce the logic gate dynamics within the smaller volume of the vesicle. To further investigate this phenomenon, a more detailed model of BRN based logic gates was created.

A more detailed model of the NOT gate gene was created based on a stochastic model of the repressilator reported by Elowitz and Leibler [79]. The repressilator is a ring oscillator built from three genes. Figure 7.7 shows a schematic diagram of the repressilator network. The system includes three different genes, LacI, λcI and TetR, with the protein expressed from each gene acting as a repressor which binds to the promoter of the next gene, and reduces the rate of transcription.

The authors present a stochastic model of the repressilator, in which all three genes and promoters have identical properties in terms of the rates of binding etc. The repression of the gene is represented by a cooperative binding of the repressor protein to the gene promoter, (Reactions 7.1 and 7.2):

$$G + R \xrightarrow{1nm^{-1}s^{-1}} GR \tag{7.1}$$

$$GR + R \xrightarrow{1nm^{-1}s^{-1}} GRR \tag{7.2}$$

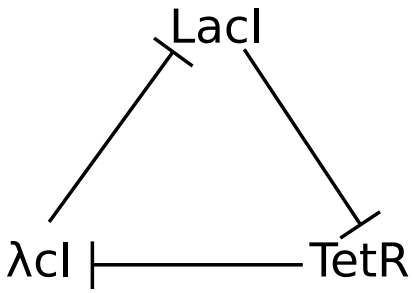


Figure 7.7: The Repressilator, the system is composed of three different genes, LacI, λcI and TetR. The protein expressed from each gene inhibits the next, so for example the LacI proteins inhibit TetR expression, and TetR proteins inhibit λcI expression.

Where G is the NOT gate promoter and gene, R is the repressor protein which binds to the gene operator and represses transcription of the gene, M is transcribed mRNA from the gene G, and O is the expressed protein from G, translated from M.

The repressor proteins also decomplex from the gene promoter, and these are modelled with reactions 7.3 and 7.4. It should be noted that the rate of decomplexation when both repressor proteins are bound to the sequence is greatly decreased when compared with the rate when only a single protein is bound.

$$GR \xrightarrow{224s^{-1}} G + R \tag{7.3}$$

$$GRR \xrightarrow{9s^{-1}} GR + R \tag{7.4}$$

Reactions 7.5 to 7.8 represent transcription and translation. Transcription when the gene promoter is unrepressed occurs 1000 times more frequently than when the gene is repressed.

$$G \xrightarrow{0.5s^{-1}} G + M \tag{7.5}$$

$$M \xrightarrow{0.167s^{-1}} M + O \tag{7.6}$$

$$GR \xrightarrow{5*10^{-4}s^{-1}} GR + M \tag{7.7}$$

$$GRR \xrightarrow{5*10^{-4}s^{-1}} GRR + M \tag{7.8}$$

mRNA and protein degradation occurs with a half-life of 120 seconds and 600 seconds respectively (Reactions 7.9 and 7.10).

$$O \xrightarrow{0.0012s^{-1}} \tag{7.9}$$

$$M \xrightarrow{0.0058s^{-1}} \tag{7.10}$$

These reactions specify a stochastic model of the behaviour of one gene in the repressilator.

If the repressing protein is considered as the input to the system, and the expressed protein the output, then the behaviour of the model mimics that of a NOT logic gate, which outputs a high signal when the input is low, and a low signal when the input is high. The gene operon has two operator regions. A repressor protein can then bind to these regions and repress the gene. When only one operator is occupied by a repressor protein, the repressor is more likely to decomplex from the gene than when both promoters are occupied, and it is this cooperative binding which causes a "switch-like" transition between the high and low output states of the logic gate [16], as a certain threshold of repressor concentration must be reached within the cell volume before both operators become occupied. The effect can be magnified by increasing the number of operators which cooperatively bind repressors, or by using oligomer proteins which must bind together before being able to bind to the gene. A stochastic P system module is defined that, using the repressilator circuitry, encodes the functionality of a NOT gate, in order to allow the construction of more complication models from this basic building block.

$$
\begin{aligned}
&NOTGate(\{R,G,M,O\},\{c_1,c_2,c_3,c_4,c_5,c_6,c_7,c_8,c_9,c_{10}\},\{b\}) = \\
&\qquad [\ G+R\ ]_b \xrightarrow{c_1} [\ GR\ ]_b \\
&\qquad [\ GR+R\ ]_b \xrightarrow{c_2} [\ GRR\ ]_b \\
&\qquad [\ GR\ ]_b \xrightarrow{c_3} [\ G+R\ ]_b \\
&\qquad [\ GRR\ ]_b \xrightarrow{c_4} [\ GR+R\ ]_b \\
&\qquad [\ GR\ ]_b \xrightarrow{c_5} [\ GR+M\ ]_b \\
&\qquad [\ G\ ]_b \xrightarrow{c_6} [\ G+M\ ]_b \\
&\qquad [\ GRR\ ]_b \xrightarrow{c_7} [\ GRR+M\ ]_b \\
&\qquad [\ M\ ]_b \xrightarrow{c_8} [\ M+O\ ]_b \\
&\qquad [\ M\ ]_b \xrightarrow{c_9} [\ \ ]_b \\
&\qquad [\ O\ ]_b \xrightarrow{c_{10}} [\ \ ]_b
\end{aligned}
$$

The module's variables $\{R,G,M,O\}$, including the continuous ones $\{c_1,\ldots,c_{10}\}$, can be instantiated with different promoters, genes, proteins and kinetic constants as to represent specific systems. Also note that the square brackets indicate that the reactions take place inside a specific (proto) membrane or compartment. Indeed to simplify the notation the compartment name variables have been removed as only one compartment is used in this study.

7.2.3 A NAND Gate

By creating two copies of the same gene, with different promoter regions, a NAND gate can be created (Figure 7.8). The NAND gate is defined by the following module. Note that the gene, mRNA and output protein are the same for both *NG* modules, but the input repressor is different

(R1 for one gene and R2 for the other).

$$NAND(\{R1, R2, G1, M1, O1\}, \{c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9, c_{10}\}, \{b\}) =$$
$$NG(\{R1, G1, M1, O1\}, \{c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9, c_{10}\}, \{b\}) \cup$$
$$NG(\{R2, G1, M1, O1\}, \{c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9, c_{10}\}, \{b\})$$

It should be noted that constructing a NAND gate in this way produces two distinct output levels when the gate output is high. In the first case when neither input to the gate is present, both genes are transcribed. However when the gate is presented with a single input one gene is repressed and the gate output, whilst still representing a logic value of True or high, produces roughly half the amount of protein than it does when no input is present.

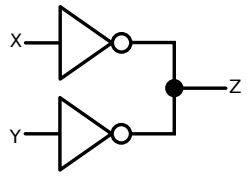


Figure 7.8: A NAND gate built from two NOT gates. The inputs to the gate are two repressor proteins labelled X and Y, and the output protein is labelled Z.

The NOT gate model can be implemented in DPD as a set of first and second order reactions. However, the rates in the original model are specified over timescales of the order of seconds, with the dynamics of the system only observable over minutes/hours. In order to observe the model dynamics within DPD timescales, the reaction rates are rescaled to occur within the DPD timescale. For the first order reactions this is a straightforward process, as long as all the first order reaction rates are scaled equivalently. However, for the second order reactions, the situation is more complex, as the stochastic rate constant is the rate at which a reactant pair will collide and react. This rate is determined by the physical properties of the system (e.g. temperature, reactant mass etc.) and the probability that the colliding particles will be in the correct orientation for the reaction to occur. Therefore to scale the second order reactions correctly, it is necessary to determine the rate at which particle pairs collide within the vesicle. The collision rate was determined directly from simulation (data not shown) and was found to be ~ 0.0002 collisions per DPD time unit. Thus the NOT gate model is instantiated as follows (where the rate constant is in DPD time units τ^{-1}).

$$NOTGate(\{P, R, G, M, O\}, \{1, 1, 1, 0.0402, 5 * 10^{-4}, 0.5, 5 * 10^{-4}, 0.167, 0.0012, 0.0058\}, \{b\})$$

For the other reactions, the rate constants were rescaled by changing the unit of time from seconds to DPD time units (τ). The consequence of this for the second order reactions representing binding of repressor to gene, is that the rate of collisions is reduced as the number of collisions per

time unit is much smaller in DPD. Therefore the actual reaction rate (R_{DPD}) is shown below.

$$R_{DPD} = [R][G]0.0002 \tag{7.11}$$

Where $[R]$ and $[G]$ indicates the number of repressors and genes in the simulation respectively. The consequence of this is that the rate at which repressors bind to the gene is reduced by a factor of 5000 in comparison with the other scaled rates. Note that the rates of the decomplexation rules which represent the repressor protein unbinding from the gene have been rescaled to $1\tau^{-1}$ and $0.0402\tau^{-1}$ (the rates in the original model were $224s^{-1}$ and $9s^{-1}$ respectively), as the original rates were too fast to be represented in the rescaling, and so were reduced by a factor of 224.

The effect of this alteration somewhat mitigates the reduction of the complexation rate, and the reduction of the binding rate relative to the adjusted decomplexation rate is reduced by a factor of 22.3. The effect of these changes will be that the repression of the gene occurs more slowly, and both the decomplexation and complexation reactions occur more slowly in comparison to the first order reactions, but the qualitative structure of the model should be maintained.

For the vesicle computation simulations of the more detailed logic gate models a vesicle was formed which was composed of 5825 DMPC molecules, encapsulating a core of 58550 solvent particles. The vesicle was then placed within a simulation space of $50r_c^3$, and the volume which was external to the vesicle membrane was filled with solvent particles such that the correct density (3 particles per r_c^3) was achieved. For each NOT gate in the module a solvent particle within the vesicle core was chosen at random and replaced with a particle representing the gene.

The effect of encapsulation on logic gate dynamics

The first experiment involved placing the NOT gate inside the vesicle membrane, and comparing the results of simulation in which the NOT gate was not encapsulated within the membrane (e.g. allowed to diffuse freely within the full $50r_c^3$ volume) to show the effect that the encapsulation has on the second order reaction rates.

The result of simulating the NOT gate within a vesicle with no input signal connected to the gate, for 5000τ is shown in Figure 7.9. The NOT gate gene is expressed when the gate has no input and the amount of protein rises until an equilibrium between expression and degradation is reached, at an output level of ~ 10000 proteins.

Figure 7.10 shows the results of simulating the NOT gate with a high input signal (i.e. a gene producing the NOT gate input repressor protein was added to the system). The gate and input gene particles were encapsulated within a vesicle and the number of expressed proteins recorded each time unit to produce a time series (continuous black line). Simulations were also performed in which the NOT gate and input gene particles were not placed within the vesicle, but instead were able to diffuse freely within the entire simulated volume (dotted black line).

At the start of the simulation the NOT gate gene is initially expressed, until the amount

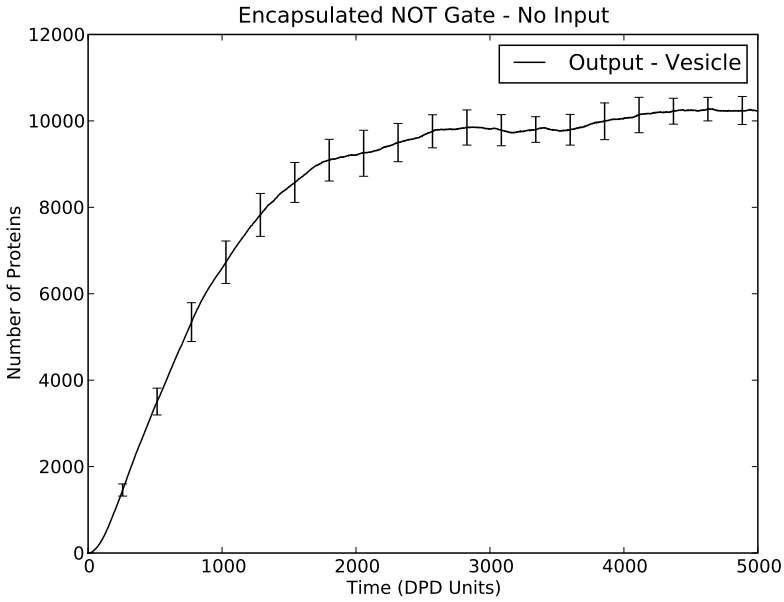


Figure 7.9: Time series of the protein output from a gene representing a NOT gate, encapsulated within a vesicle showing the mean number of proteins present in the volume against simulated time in DPD units (each DPD time unit is approximately 88ps). The error bars showing the estimated standard error.

of repressor (which is concurrently expressed from the input gene) reaches the threshold required to fully repress the NOT gate gene, causing the amount of output protein to drop as the protein and mRNA degrade and are not replenished, so that typically less than 50 proteins remained by the end of the simulation.

The dotted line in Figure 7.10 shows the result of simulating the high input model for the case were the input and NOT gate genes were not encapsulated within the vesicle. The mean number of proteins expressed at the peak of expression was greater by $\sim$ 1000 particles when compared to the output time series for the encapsulated gate, and the peak was reached later in the simulation, indicating the transition of the NOT gate (module NG) from the high to low output state occurred more slowly. The mean time series for the input repressor protein in the encapsulated and unencapsulated NOT gate simulations are shown by the grey continuous and grey dotted lines respectively. Once the repressor protein levels have reached an equilibrium, there is a difference of over $\sim$ 1000 proteins between the encapsulated and unencapsulated equilibrium value. Correspondingly there is a difference between the levels of outputted mRNA when the NOT gate was and was not encapsulated, Figure 7.11 shows that the mean mRNA output when the gate was not encapsulated peaked at slightly less than 60 molecules, whereas the mRNA output for the encapsulated gate peaked at around 43 molecules. As the collisions occur more frequently in the vesicle volume, the gene becomes fully repressed more quickly, and so the peak level of mRNA output

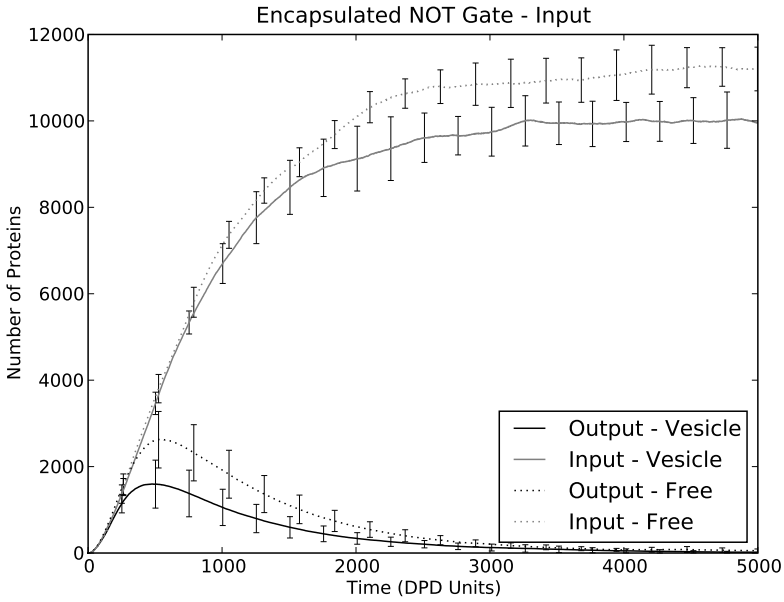


Figure 7.10: Output protein levels for simulation of NOT gate placed within a vesicle and diffusing freely in the simulated volume, averaged over 10 runs (error bars indicate estimated standard error). The continuous grey and black lines show the time series for the input and output proteins for the NOT gate placed within a vesicle. The dotted grey and black lines show the input and output proteins of the NOT gate with an input present when system is not encapsulated within a vesicle.

is reduced.

Figure 7.12 shows the output protein levels from the NAND gate model built from two NOT gates (model NAND in section 7.2.3). Four time series are shown, one for each possible combination of signal inputs to the gate. The continuous line shows the output for the gate when both of the genes for the input signals (labelled X and Y in the figure) were present. The output level rises initially until enough of the input proteins are present to fully repress both genes in the NAND gate, at which point the output signal drops to zero. The dotted line shows the case where there was no input signal to the NAND gate and the level of output protein reaches an equilibrium value of around 17500 proteins. The dashed and dot-dashed lines show the case where one of the input signal genes was present. In both of these cases, one of two NOT gates which make up the NAND is repressed, and so the output protein levels reaches an equilibrium value of around 8000 proteins, which is roughly half the output level when neither input was present.

7.3 P System Specification of Compound Liposome Logic Models

In this section some compound models (i.e. composed from a hierarchical combination of modules) of more complicated liposome logic circuits are expressed as P systems specifications. Models of logic gate based systems which have dynamics that occur over timescales too long for DPD, or

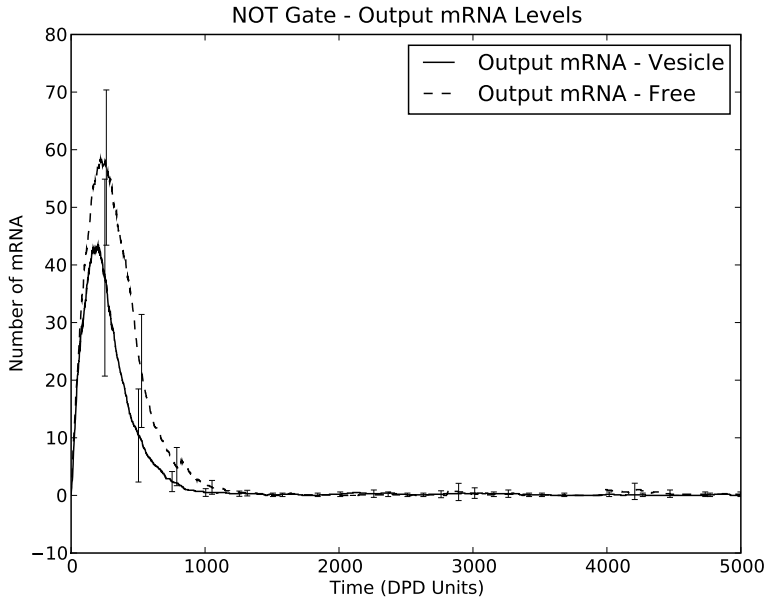


Figure 7.11: The time series for the transcribed mRNA from the NOT gate gene, averaged over 10 runs. The continuous black line shows the output level of transcribed mRNA when the NOT gate was encapsulated within the vesicle, whereas the dashed line shows the mRNA time series when the NOT gate was diffusing freely throughout the entire volume.

are composed of large numbers of logic gates are better simulated using the stochastic simulation algorithm.

7.3.1 A P System Specification for the Repressilator

Logic gates in cellular computing are constructed from networks of gene regulation in prokaryotic genomes. In prokaryotes, genes are sometimes arranged into operons, sequences of DNA containing a promoter region which is recognised by RNA polymerase enzymes, an operator region which is recognised by gene transcription factors, and one or more gene sequences (see Figure 7.13). To construct the P system module representing the repressilator, an instantiation of the *NOTGate* module named *NG* is derived so as to avoid specifying the stochastic rate constants each time (which are the same unless otherwise specified):

$$NG(\{R, G, M, O\}, \{\}, \{b\}) =$$
$$NOTGate(\{R, G, M, O\}, \{1, 1, 224, 9, 5 * 10^{-4}, 0.5, 5 * 10^{-4}, 0.167, 0.0058, 0.0012\}, \{b\})$$

Three or more *NG* modules can be connected together in sequence, with the output of the last connected as the input of the first gate, to produce a ring oscillator. Ring oscillators made from

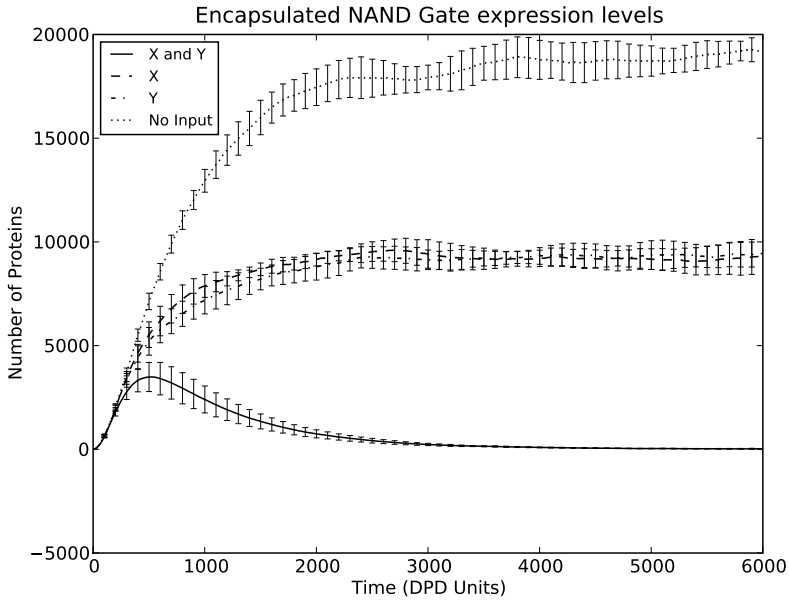


Figure 7.12: The time series of protein output levels from the simulation of the NAND gate, the output of the gate is shown in response to 4 different combinations of inputs labelled X and Y

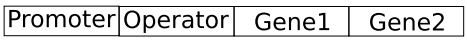


Figure 7.13: The operon in prokaryote genomes, the promoter region is recognised by RNA polymerase, which binds to the promoter to initial transcription. The operator is recognised by transcription factor proteins which alter the rate of gene expression, the operator may control the expression of multiple genes.

N gates can be constructed as follows. For any odd integer N greater than or equal to three:

$$
\begin{aligned}
RON(\{G_1, \cdots, G_N, M_1, \cdots, M_N, O_1, \cdots, O_N\}, \{\}, \{b\}) = \\
& NG(\{O_N, G_1, M_1, O_1\}, \{\}, \{b\}), \\
& NG(\{O_1, G_2, M_2, O_2\}, \{\}, \{b\}), \\
& \cdots \\
& NG(\{O_{N-1}, G_N, M_N, O_N\}, \{\}, \{b\})
\end{aligned}
$$

Therefore when $N = 3$ the original Repressilator model, named RO3, is reproduced.

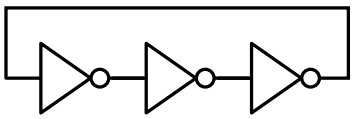


Figure 7.14: A ring oscillator built from three not gates.

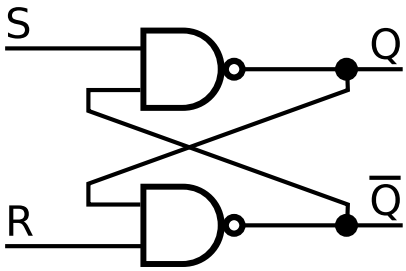


Figure 7.15: Set Reset Latch constructed from two NAND gates.

7.3.2 A Set-Reset Latch

Two NAND gates can then be connected to create a Set-Reset Latch (Figure 7.15). The output of each gate is connected to the input of the other, and the state of the latch can be switched by holding the remaining set or reset inputs high for a short period. The Latch acts as a simple one bit memory which can be set or reset by expressing the appropriate protein that represses the gene of the relevant NAND gate. The Latch module is built from two NAND gates

$$\begin{aligned} SR-Latch(\{R1,R2,G1,G2,O1,O2\},\{\},\{b\}) = \\ NAND(\{R1,O2,G1,M1,O1\},\{\},\{b\}) \cup \\ NAND(\{R2,O1,G2,M2,O2\},\{\},\{b\}) \end{aligned}$$

7.3.3 A D type Flip-Flop

Latches can then be connected to NAND and NOT gates to construct a D type flip-flop, as shown in Figure 7.16.

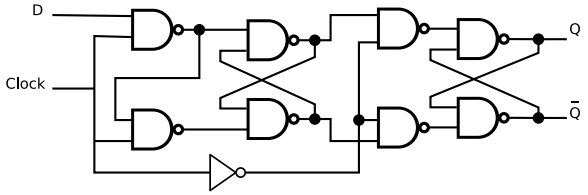


Figure 7.16: The D Flip-Flop built from two latches, four NAND gates and a NOT gate.

A D Flip-Flop takes a data input, indicating whether the flip-flop should be set or reset, and a clock input. The output of the flip-flop will be the last active input when the clock was still high, and so the output is fixed when the clock input goes from high to low. Each flip-flop stores a single bit, and can be coupled in sequence to make larger memories. The D-Flip Flop can also be converted to a toggle flip-flop by connecting the $\bar{Q}$ output to the D input. Therefore each time a clock pulse occurs, the gate will toggle between the Set and Reset states. The module configuration

for the D flip flop is shown below:

$$DFlipFlop(\{G1,\cdots,G9,DInput,ClockInput,M1,\cdots,M8,O1,O2\},\{\},\{b\}) =$$
$$NG(\{ClockInput,G9,M9,O9\},\{\},\{b\})\cup$$
$$NAND(\{Dinput,ClockInput,G5,M5,O5\},\{\},\{b\})\cup$$
$$NAND(\{O5,ClockInput,G6,M6,O6\},\{\},\{b\})\cup$$
$$SR-Latch(\{O5,O6,G1,G2,O1,O2\},\{\},\{b\})\cup$$
$$NAND(\{O1,O9,G7,M7,O7\},\{\},\{b\})\cup$$
$$NAND(\{O9,O2,G8,M8,O8\},\{\},\{b\})\cup$$
$$SR-Latch(\{O7,O8,G3,G4,O3,O4\},\{\},\{b\})$$

7.3.4 A 3 bit Ripple Counter

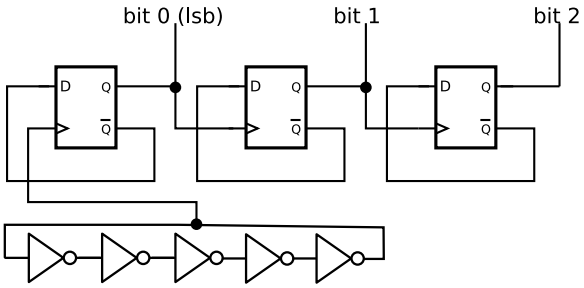


Figure 7.17: A 3 bit counter connected to a 5 gate ring oscillator.

A toggle flip-flop can in turn be used to build ripple counters, simple counters in which the the Q output of one flip flop is connected to the clock input of the next flip flop. Figure 7.17 shows a diagram of three flip flops connected together to form a 3 bit ripple counter, with a 5 NOT gate ring oscillator acting as the system clock. When the clock is high, the state of the first flip-flop is toggled to produce a high output, connected to the clock input of the next flip-flop, which is then also toggled. The first flip-flop remains in the logic high state until the next clock pulse, upon which it toggles to the low state, causing the state of the second flip-flop to be fixed high. As the output of each flip flop toggles at half the rate of it clock input, the output of the first flip flop is high for one clock cycle, and then low for one clock cycle. The second bit high for two cycles and low for two cycles, and the third bit high for four cycles and low for four cycles.

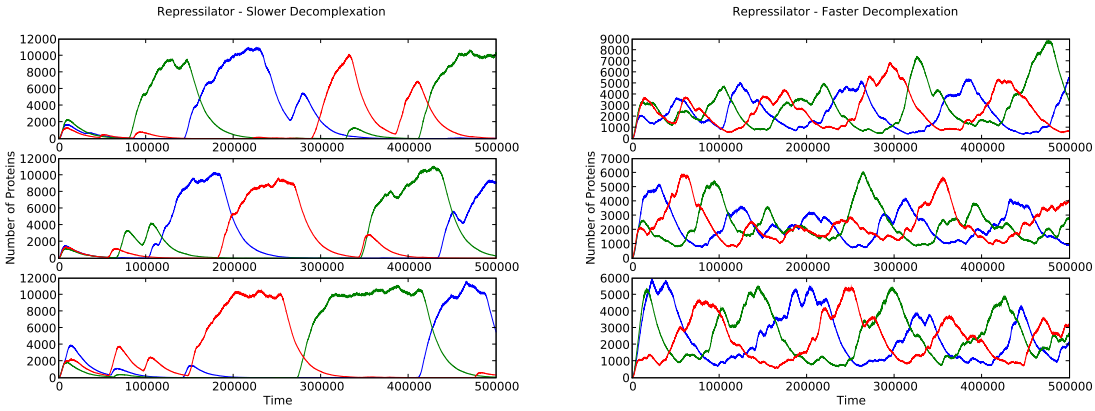


Figure 7.18: Simulations of the repressilator model within a vesicle in DPD, the figure on the left shows time series from three different simulations of the repressilator with reaction rate parameters which are rescaled versions of those from the Elowitz model such that the dynamics can be examined within DPD timescales. The figure on the right show three time series from simulations of the same model, but with increased rate constants for the decomplexation of the repressors form the promoter.

The counter module is constructed from the following modules:

$$
\begin{aligned}
Counter-3bit(\{G1,\cdots,G28,M1,\cdots,M35,O1,\cdots,O8\},\{\},\{b\}) = \\
DFlipFlop(\{G1,\cdots,G9,ClockInput,O2,M1,\cdots,M9,O1,O2\},\{\},\{b\})\cup \\
DFlipFlop(\{G10,\cdots,G18,O1,O4,M10,\cdots,M18,O3,O4\},\{\},\{b\})\cup \\
DFlipFlop(\{G19,\cdots,G27,O3,O6,M19,\cdots,M27,O5,O6\},\{\},\{b\})\cup \\
5GateClock(\{G28,\cdots,G35,O5,O8,M28,\cdots,M35,O7,O8\},\{\},\{b\})
\end{aligned}
$$

7.4 Repressilator Simulations in DPD

Time series from simulations of the increased decomplexation rate repressilator model, encapsulated within a vesicle are shown in Figure 7.18. The expressed protein levels for each of the three NOT gates in the repressilator (shown for for three runs of the simulation) can be seen to oscillate. The increased decomplexation rate of the repressors from the gene when compared to the original repressilator model means that the period of oscillation is not quite long enough to allow all of the transcription factor to degrade, and so the amount of each transcription factor drops to around 1000 proteins.

Figure 7.18 shows the results of simulating the model where the decomplexation rates were only scaled, and not increased. The decomplexation of repressor from gene occurs less frequently in this model and so the oscillations have a longer period, allowing the transcription factors to degrade completely before the next cycle of the oscillation and the period of the oscillation is increased.

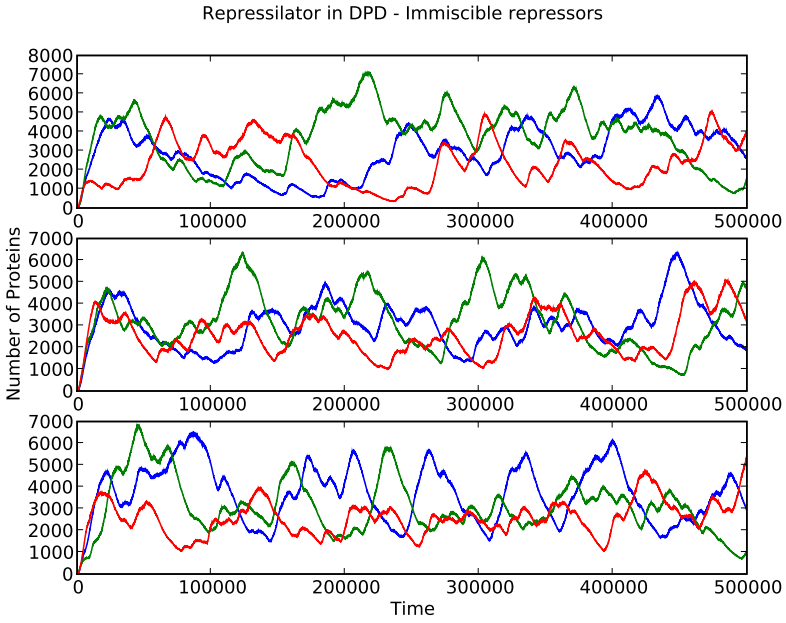


Figure 7.19: Simulations of the repressilator model with hydrophobic repressor proteins. The figure shows time series from three different simulations.

The figure also shows the results of simulating the repressilator model where the rate constants for decomplexation have been increased. Figure 7.20 shows the images from the inner volume of vesicle, with the particles representing the different output proteins from each of the three NOT gates given different colours, each of the images are captured at the point in the simulation were the respective protein is being expressed.

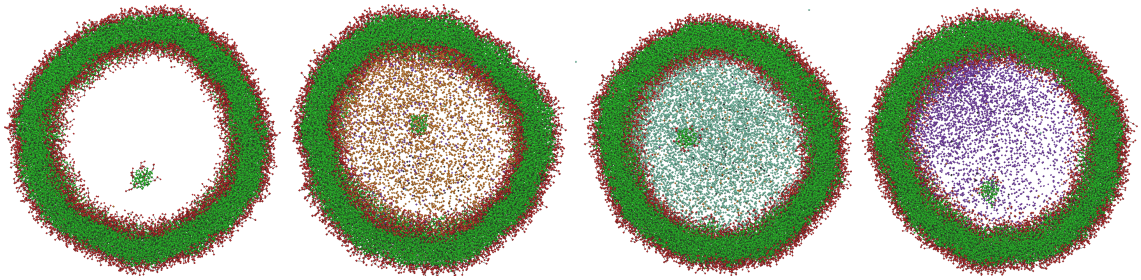


Figure 7.20: Snapshots from a simulation of the repressilator within a vesicle were taken every 2500τ, the vesicle membrane is composed of hydrophobic tail chains (green) and hydrophilic head groups (red), a small micelle was trapped within the vesicle when it formed and is visible in each image. The vesicle was sliced so that the inner volume is visible (note that solvent particles are not shown). The images show (from left to right) the initial vesicle condition, high concentrations of the output protein expressed from the first NOT gate, the second NOT gate, and the third NOT gate (note the concentration gradient visible in the last image).

	Solvent	Gene	R1	R2	R3
Solvent	78	78	85	85	85
Gene	78	78	78	78	78
mRNA	78	78	78	78	78
R1	85	78	78	85	85
R2	85	78	85	78	85
R3	85	78	85	85	78

Table 7.3: α parameters for immiscible repressors.

Immiscible Repressors

The third vesicle computing experiment involved the same initial configuration as the previous repressilator experiment (module RO3), but the alpha parameters for the proteins were modified slightly to examine the case where the output protein is slightly hydrophobic, and also less miscible with other proteins. The effect of this should be to create three distinct protein phases, which may mean the dynamics of the repressilator will be altered due to the non-uniform concentrations of repressors. Table 7.3 shows the alpha parameter vector for each repressor protein in the system.

The results from simulations of the repressilator with increased α parameters between the repressor proteins expressed from each NOT gate gene are shown in Figure 7.19. Because the transcription factors are now hydrophobic and do not mix with the solvent, the volume is no longer homogeneous, causing the dynamics of the repressilator to be altered. The period of the oscillations is no longer steady as the gene might not diffuse into an area that contains a high concentration of proteins that repress it. The repressor proteins also form distinct phases which tend to move towards the boundary between the vesicle membrane and the solvent, so that contact between hydrophobic repressor and solvent is minimised. The result of this movement was a bulging deformation of the normally spherical vesicle shape (shown in Figure 7.21). Deformation of the membrane may be interesting to those working on the problem of causing vesicle fusion, as the deformation of the membrane will create areas of increased tension due to the elasticity of the membrane, which may increase the likelihood of fusion if two such vesicles were to come into close contact [232]. This result also illustrates the sort of system dynamics that can be observed in DPD rather than in other less detailed simulation techniques.

7.5 Stochastic Simulation Algorithm Results

If more complex logical circuits need to be simulated, or simulations for long length/timescales are required, the molecular and three dimensional detail of DPD can be abstracted away, and a stochastic simulation algorithm can be used instead to simulate deeper logic circuits that capture compartments' topologies but ignores their detailed geometries. The results of the SSA experiments are now described.

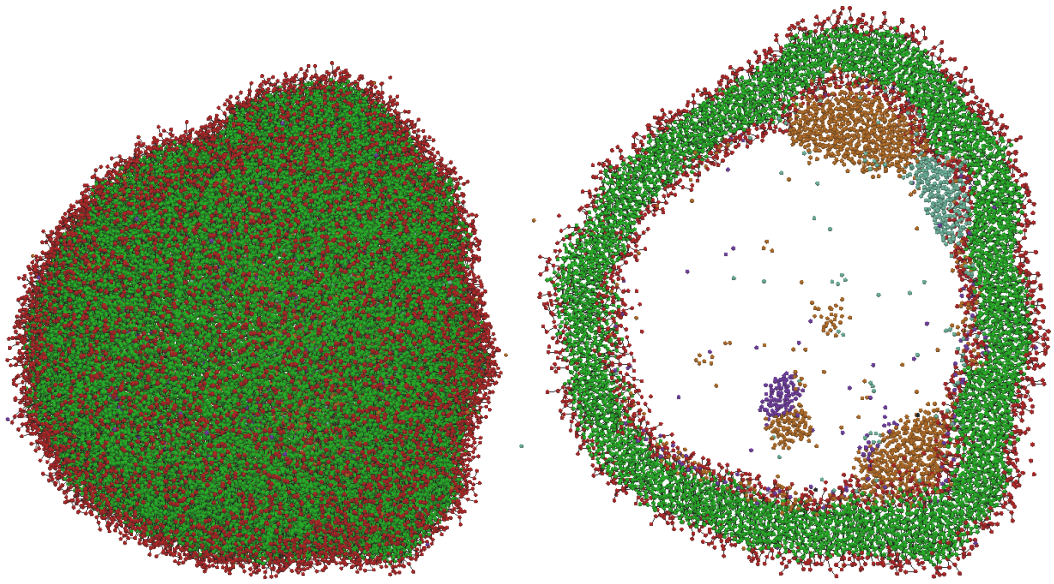


Figure 7.21: Hydrophobic repressor domains form within the vesicle, and deform the membrane: The image on the left shows the surface of a vesicle which has been deformed by the formation of phases within it. The image on the right shows a slice through the same vesicle, the output proteins (coloured orange, blue and purple) have formed phases in the vesicle core and are pressing against the membrane.

Oscillator Frequency

The Elowitz models were extended with increasing numbers of NOT gates, to investigate whether increasing clock periods would match the theoretical estimates for silicon gates, and if there are limits to the number of gates which can be connected together in this way. The oscillator models were constructed from 5,7,9,11,21,31,41 and 51 gates modules (RO3,RO5,etc.) and simulation of each oscillator was performed for 2 days of simulated time.

The formula for calculating the frequency of a electronic ring oscillators built from any odd number of NOT gates is shown in Eq. 7.12:

$$\frac{1}{2nTp} \tag{7.12}$$

Where n is the number of logic gates, and Tp is the propagation delay of each gate. To determine if this formula accurately calculates the frequency of the oscillators built from logic gates the propagation delay was calculated for the gates, and the oscillator frequency was calculated from the output data, which was then compared with the values calculated from the formula.

The propagation delay of the NOT gate was determined to be 766.46±1.95 seconds, by simulating an NOT gate with the initial number of input repressor proteins set to the mean equilibrium output for the gate (11983±47.29), with a constant input of repressor protein also present. The propagation delay was determined as the mean number of seconds for the NOT gate output to

fall to half of its original level. The results from simulation of oscillators with 1,3,5,7,9,11,21,31,41 and 51 NOT gates are shown in Figure 7.22. The relationship between the number of NOT gates and oscillator frequency is similar to equation 7.12 until the number of NOT gates is 11 although the frequency is reduced by between 0.3 and 1 microhertz. For oscillators with more than 11 NOT gates the standard deviation of the frequency is increased, and the shape of the curve no longer follows the predictions from equation 7.12. Looking at the data for each individual run showed that for 21 NOT gates and above, the oscillator was decreasingly likely to settle into a stable oscillation. Table

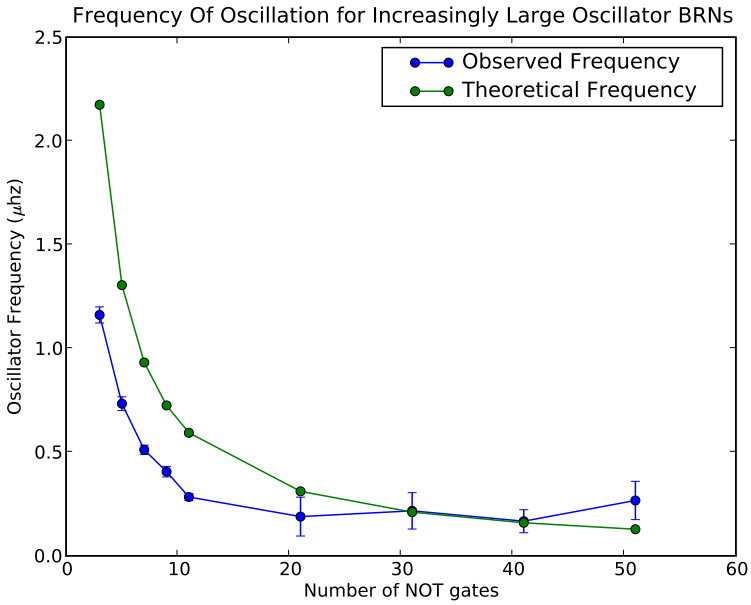


Figure 7.22: The figure shows the frequency of oscillation in μhz for oscillators constructed from 1,3,5,7,9,11,21,31,41 and 51 NOT gates. The blue line shows the the oscillator frequencies observed in simulation, each point is the mean frequency of 10 simulations of the oscillator, the error bars show the standard deviation. The red line shows the frequency calculated from equation 7.12 for the different numbers of gates.

7.4 shows the number of oscillators in the 10 runs which were unstable for the different numbers of NOT gates.

The effect of RNAP and Ribosomes

The behaviour of the RO51 oscillator was also examined in a more detailed model where the transcription and translation explicitly included polymerase and ribosomes. The number of polymersomes and ribosomes were at realistic levels for a bacterial or large vesicle volume.

When RNAP and ribosome interactions are included explicitly in the model, the effect is that there is a global constraint on the rate of transcription and translation. Figure 7.23 shows the levels of free RNAP and ribosomes for a simulation of the 51 gate oscillator model modified

Number of NOT Gates	Unstable Count
3	0
5	0
7	0
9	0
11	0
21	1
31	5
41	7
51	9

Table 7.4: The number of unstable oscillations observed during 10 runs of oscillators composed of different numbers of NOT gates.

to include RNAP and ribosome interactions explicitly. The model was initialised with 35 RNAP and 350 ribosomes. The result shows that when the oscillator is functioning the average number of RNAP in use is slightly less than one, and the average number of ribosomes in use is around 25. However the inclusion of the RNAP and Ribosomes did not alter the transcription rate significantly.

The 3-bit Ripple Counter

The counter models were simulated in MCSS for simulated time periods of either 2 or 3 days, with the number of molecules of each chemical species recorded at every 3 minutes of simulated time to produce a time series for each chemical species in the simulation.

Figure 7.24 shows the time series for the simulation of the 3-bit counter with 3 and 5 gate ring oscillators as the clock. The time series show the output protein levels for each bit of the 3-bit counter. In the case of the counter connected to a 3-gate clock, it is likely that the propagation delay of the flip flops is greater than the time between clock pulses, and so the output of the first counter bit (proteinG18) does not always indicate that the flip flop was correctly toggled by the clock input. When the counter is connected to a lower frequency clock (constructed from 5 NOT gates), the dynamics of the output of the first counter bit have a much more consistent period and number of period of high output is roughly 1/2 the number of input clocks as expected. Figure 7.25 shows the clock input and first bit output overlaid for the 5 gate clock model. The figure shows that there is a clear correspondence in each case between the high level of each bit and the triggering of the output of the next bit, the counter is therefore functioning as intended. Note that when the counter reaches its limit (7 in this case) it simply overflows and the counter starts from zero again.

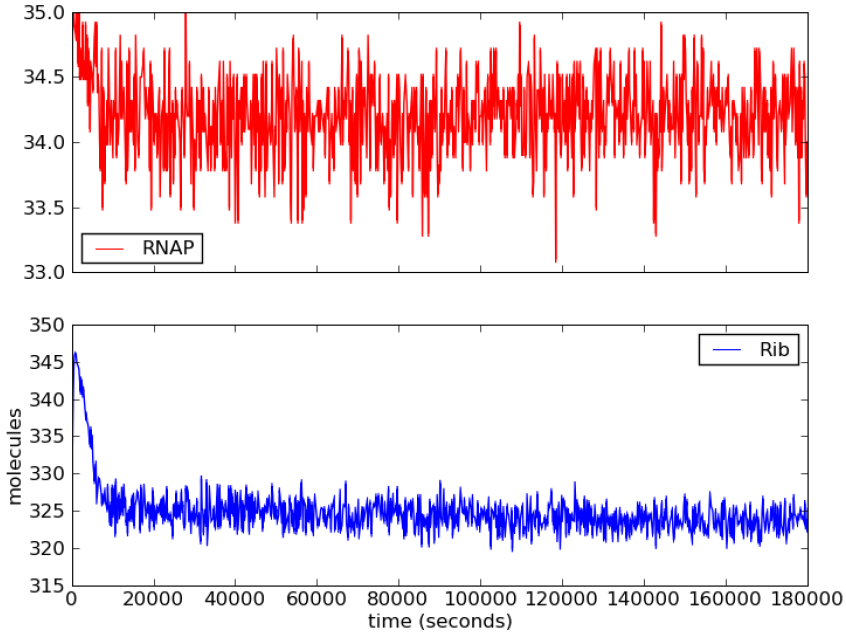


Figure 7.23: Time series for free RNA polymerase (RNAP) and Ribosome (Rib) proteins in a simulation of the 51 gate oscillator model.

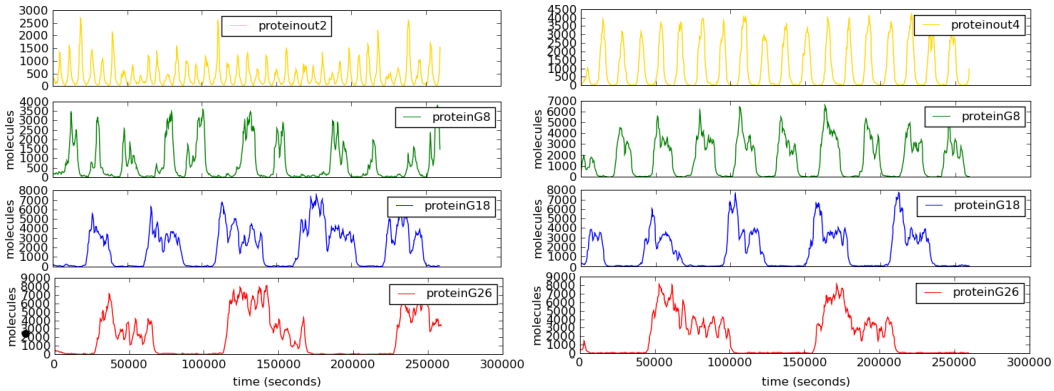


Figure 7.24: Time series for 3-bit counter model with 3-gate and 5-gate clocks as input, for the 3-gate clock (left) **proteinout2** is the clock signal, **proteinG8** is the output of the first bit of the counter, **proteinG18** the output of the second bit of the counter and **proteinG26** is the output of the third bit. For the 5-gate clock (right) **proteinG8** is the output of the first counter bit, **proteinG18** the output of the second bit of the counter and **proteinG26** the output of the third bit.

7.6 Model Checking

The analysis is focussed on two of the simplest parts in this study, namely the NOT gate and the NAND gate, that are subsequently used to construct the rest of the models. In order to assess their performances formal analysis was applied to their dynamics using *simulative probabilistic model checking*. More specifically, the behaviour of the P system models were translated into CTMCs

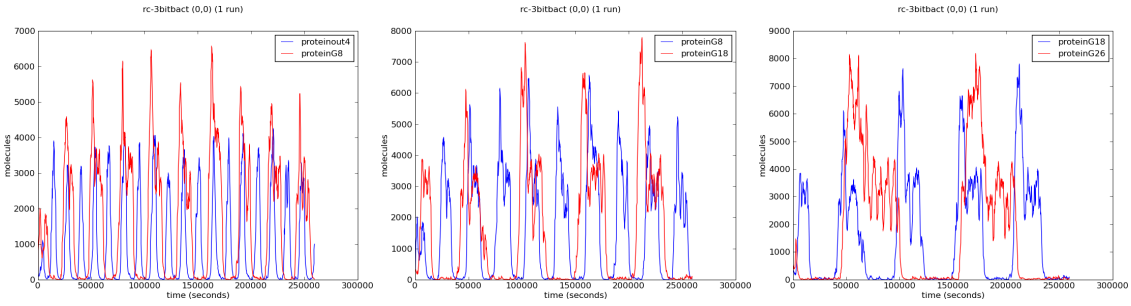


Figure 7.25: Overlaid time series for protein output levels, the top figure shows the clock input level overlaid with the bit-0 output for the counter, the middle figure shows the bit-0 output overlaid with the bit-1 output, and the bottom figure shows the bit-1 output overlaid with the bit-2 output.

and then analysed using the PRISM probabilistic model checker [148]. Due to the complexity of the models under study the complete state space was not constructed, but, instead, ensembles of multiple simulations or trajectories in the state space were generated and the corresponding properties, expressed in the temporal logic CSL [148], were checked against them.

In the analysis that follows 1000 simulations were used to produce an estimate $\hat{p}$ of the answer p to a query. This resulted in a *precision* of 0.1 with a *confidence* of 0.01 which determines the accuracy of the estimate according to the following formula.

$$P[\ |p - \hat{p}| > precision\ ] < confidence$$

7.6.1 NOT Gate

In the case of the molecular NOT gate the accuracy of its behaviour was studied with respect to the general specification of a NOT gate and the speed of its response when provided with some input molecules.

Expected number of output proteins in the long run for different values of input proteins.

The basic NOT gate building block was examined using model checking to determine if it behaves as expected. That is, in the presence of low values of input proteins, high levels of output proteins should be produced and vice versa, when high amounts of input proteins are provided, no output protein should be synthesised.

In order to investigate this, the following *instantaneous reward formula* was formulated and a reward corresponding to the number of output proteins was associated to each state in the corresponding continuous-time Markov chain.

```
R = ? [ I = 6000 ]
```

The property was analysed at the time instant I = 6000 seconds. Figure 7.26 shows that for low numbers of CI proteins the number of output proteins in the long run is high, whereas an

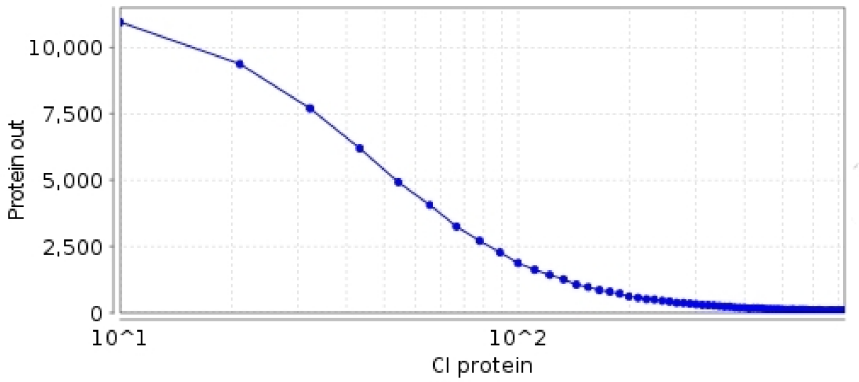


Figure 7.26: Expected number of output proteins for different number of initial input proteins in the NOT gate (logarithmic scale).

increase in the number of input proteins produces a sharp decrease to zero in the number of output proteins. The transition from high to low output occurs at around 150 input proteins. These results are in agreement with the general specification of a NOT gate.

Expected propagation time or response time.

The time taken for the molecular device to respond to its input was investigated by determining the time expected to reach half way between the initial and the final state once input proteins are introduced in the system. This property is normally termed *propagation time* or *response time*.

The following *reachability reward formula* was considered in order to investigate the propagation time of the NOT gate.

```
R = ? [ F proteinOut < 5000 ]
```

This type of query accumulates, over a trajectory, the rewards associated with each state times the time spent in that state until a state fulfilling the corresponding formula is reached. Since the aim is to accumulate the time spent in each state over a given trajectory a reward equal to one is associated to each state in the corresponding CTMC.

The property whose reachability needs to be analysed is the output protein descending below the threshold of 5000 molecules, which is half of the the initial number of output proteins, 10^4 molecules. In Figure 7.27 we can observe that a low number of input proteins leads to a very slow response, whereas an increase in the number of input molecules produces a fast decay in the propagation time. Interestingly, the study shows the existence of a threshold for the input proteins around 150 for which any further increase does not produce an acceleration in the response.

From these two properties it can be concluded that for the NOT gate there exists a threshold of around 150 input proteins. Below this number the molecular device produces a high number of output proteins, if a number of input proteins above this threshold is provided to the system then

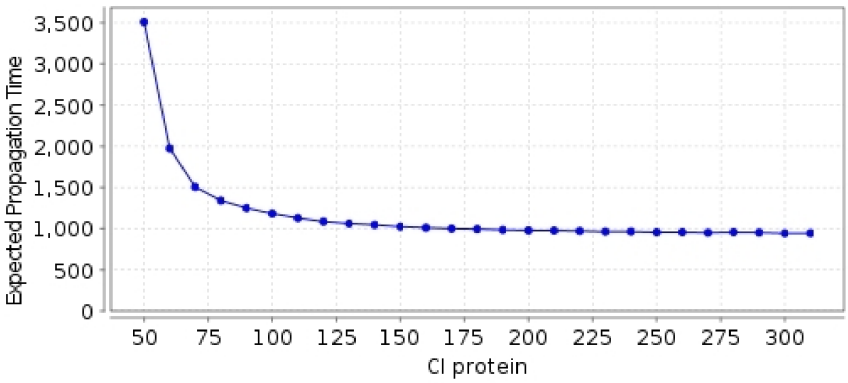


Figure 7.27: Expected propagation time for the NOT gate with different number of initial input proteins.

no output proteins are synthesised. Moreover, this threshold of 150 proteins provides the optimal input value with respect to the propagation time, as an increase in the input beyond this level does not produce a faster response.

7.6.2 NAND gate

Similar to the previous case for the NOT gate, properties are studied to determine the accuracy of the behaviour of the genetic design when compare to the general specification of a NAND gate.

Expected behaviour of the NAND gate.

In the presence of both inputs the molecular device should synthesise no output proteins whereas in any other case, that is, presence of only one input or absence of both inputs, output proteins should be detectable.

The following *instantaneous reward property* is used to determine the number of output proteins in the long run, time instant I = 6000, for different values of the two input proteins.

```
R = ? [ I = 6000 ]
```

Note that since the NAND gate is a composition of two identical NOT gates with the same parameters as the one analysed above the threshold of 150 input molecules is also evident in the behaviour of this gate, Table 7.5. This determines four different regimes in the behaviour of the gate. When $INPUT_1$ and $INPUT_2$ are less than 150 the output is maximal. When $INPUT_1$ is less than 150 and $INPUT_2$ is greater than 150 (similarly when $INPUT_2$ is less than 150 and $INPUT_1$ is greater than 150) the output is produced at a half maximal level. Finally, no output proteins are synthesised when both $INPUT_1$ and $INPUT_2$ are greater than 150.

X Input (Proteins)	Y Input (Proteins)										
	0	50	100	150	200	250	300	350	400	450	500
0	11987	8381	6970	6472	6296	6176	6124	6049	6071	6092	6091
50	8475	4862	3437	2945	2795	2659	2575	2559	2545	2495	2501
100	6962	3412	1896	1435	1243	1125	1086	1076	1029	969	987
150	6485	2942	1437	949	749	645	599	596	547	525	540
200	6305	2764	1248	752	581	470	424	402	359	364	344
250	6186	2642	1168	669	477	388	327	291	261	249	232
300	6113	2607	1080	610	422	339	262	245	211	189	187
350	6011	2586	1045	566	378	313	235	192	186	164	143
400	6110	2577	1019	552	348	269	209	175	161	140	133
450	6113	2513	1031	546	343	243	196	149	143	127	122
500	6122	2489	1028	539	332	251	189	142	138	109	106

Table 7.5: Expected number of output proteins in the long run for the NAND gate with different numbers of input proteins.

X Input (Proteins)	Y Input (Proteins)										
	0	50	100	150	200	250	300	350	400	450	500
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
50	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
100	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
150	0.0	0.0	0.0	0.0	0.03	0.12	0.18	0.25	0.35	0.26	0.49
200	0.0	0.0	0.0	0.02	0.25	0.64	0.76	0.82	0.92	0.92	0.95
250	0.0	0.0	0.0	0.08	0.55	0.91	0.96	1.0	0.98	0.99	1.0
300	0.0	0.0	0.0	0.23	0.81	0.93	1.0	1.0	1.0	1.0	1.0
350	0.0	0.0	0.0	0.25	0.81	0.99	0.99	1.0	1.0	1.0	1.0
400	0.0	0.0	0.0	0.4	0.84	0.99	1.0	1.0	1.0	1.0	1.0
450	0.0	0.0	0.0	0.36	0.95	1.0	1.0	1.0	1.0	1.0	1.0
500	0.0	0.0	0.0	0.51	0.98	1.0	1.0	1.0	1.0	1.0	1.0

Table 7.6: Probability of the absence of a detectable level of output proteins from the NAND gate for different levels of both inputs.

Probability of the absence of a detectable level of output proteins.

In order to get a more detailed intuition of the behaviour of the NAND gate we estimated the probability of a non-detectable level of output proteins in the long run for different values of the two input proteins. The detectable level was fixed to 500 output proteins. For this the following *transient probability formula* was used.

```
P = ? [ true U[6000,6000] proteinOut < 500  ]
```

Table 7.6 shows the sharp transition around the threshold of 150 input proteins from a detectable level of output proteins to an undetectable one.

7.7 Conclusions

In this chapter the investigations of the reaction dynamics of a model vesicle computing system constructed from simple boolean logic gates were presented. The input to this pipeline is a formal specification of the vesicle computing model in stochastic P systems. This formal specification language is independent of the simulation paradigm used to study the dynamics of the model. Specifically, in this work I have used DPD and SSA to simulate the behaviour of the models. Moreover, the P system specifications enable automatic reasoning, with model checking, of systems and synthetic biology designs at a high level of abstraction.

Modularity in P system models allowed the development of models in a parsimonious manner. Initially a model of a NOT logic gate was developed, based on the rates and reactions specified in Elowitz's stochastic model of the repressilator. The expressive power of the P systems specification was then illustrated by combining NOT gate modules to make NAND gates, which were in turn used to create a Set-Reset latch, which formed the basic component required to create more complicated flip-flop and counter modules.

In principle any applicable simulation technique could then be chosen to investigate the models. In this chapter the focus was on the analysis of the logic gate models at a very high level of detail in DPD, enabling qualitative understanding to be gained about the behaviour of the model logic gates when encapsulated within a self-assembled liposome. Simulations of these models indicated that encapsulation within the liposome had the effect of increasing the rate of reaction. The SSA method was chosen to explore the behaviour of models which would be too computationally expensive to investigate with DPD, and simulations using this technique showed that complicated logic designs such as a 3-bit counter functioned correctly for several days of simulated time.

Designing a BRN using the simulation framework might involve several iterations of the model specification and simulation phases, with non-working prototypes reworked after each iteration. Once the simulations indicate a working design, the designer can move to the next stage in the pipeline, and perform a more robust analysis into the behaviour of the design using model checking. For our liposome logic designs, model checking was performed on the NOT and NAND gates, and temporal logic CSL queries used to determine the propagation delay and minimum input for the gates.

The proposed simulation framework was used to illustrate and investigate the concept of performing simple computation in liposomes or vesicles rather than biological cells. The benefit of this approach is that starting from the bottom up and creating systems containing only the required functionality will likely make the system more predictable. The vesicle membranes could also act as a barrier between the implementation of a logic component and the external environment. This provides a second level of modularity as computing systems could be built as a collection of vesicles encapsulating modules that only interact through a well defined interface, the membranes.

Three simple logic gates were implemented and the effects of encapsulation on their dy-

namics was investigated using a new method based on DPD simulations. The results showed that by constraining the gene transcription/translation molecules within the liposome, the response time of the gate (i.e. the time for the gate to produce an output after a change of input) was reduced drastically by ensuring a high concentration of activators/repressors within the liposome inner volume. This ensures that the activator and gene are in close proximity, so the activator does not have to diffuse very far to collide with the gene after being activated by a signal molecule. Clearly, simple logic gates are not useful in isolation, and so a key area of further research is to investigate methods by which the liposome gates could be combined into more complex functionality. Several possibilities spring to mind. One liposome logic gate could be nested within another, with the inputs and outputs of the liposomes coupled together, or the gates could be held together by molecular tethering to ensure the output of one gate is not dispersed before reaching the next gate. By synthesising simple protocell like structures containing selected biochemistries, a much wider range of bio and standard chemistries are available as building blocks for computation, and are less likely to suffer from incompatibilities with complex biological systems in a heterogeneous environment. With regard to synthetic biology, combination of these structures in relation to existing biomolecular machinery could allow the modular design and specification of artificial life from the bottom up.

In the next chapter, the effect of chemical communication is investigated over populations of vesicle computing devices using the lattice population P systems technique.

Chapter 8

A Solver for instances of the 2-SAT problem

This section illustrates the application of the vesicle computing simulation framework for the specification, execution and analysis of multicellular systems. In this case a solution of the 2-SAT problem using engineered populations of bacteria or vesicles. Models are executed with a stochastic simulation algorithm to compute the dynamics of the system under study. Properties of these models can be expressed using temporal logics and rigorously analysed using probabilistic model checking.

8.1 Introduction

In this chapter the lattice population P systems functionality of the modelling framework is used to design a solution to the 2-SAT problem, the Boolean satisfiability of formulas in conjunctive normal form containing a maximum of two different variables. These simulations illustrate the manner in which a distributed, amorphous style computation might be performed using chemical signalling and vesicle computing devices. The resulting models are general enough to apply to both vesicle and cellular computing implementations, although are perhaps more appropriate as vesicle computing models as the model cells are assumed to contain only the designed spatial 2-SAT gene regulatory networks (GRNs). The solutions are constructed using synthetic gene regulatory networks that represent propositional logic formulas and spatial patterns of signal molecules to codify truth assignments to the variables of the formulas. Although 2-SAT is not formally a hard problem and notwithstanding that NP-hard problems have been tackled with synthetic biology, focussing on the 2-SAT problem allows the further exploration of the merits and potential of the integrative methodology described in previous chapters, and provides a good test case for the extension of the vesicle computing simulation and modelling framework to whole colonies of vesicle computing devices.

Analysis of mRNA sequences has determined that naturally occurring motifs in the *Escherichia coli* genome [239] can be shown to function in the same way as logic gates. Creating synthetic networks using the logic gate approach is also a common theme in synthetic biology [264, 262, 16, 263, 142, 188, 100, 114, 121, 200]. If the properties of electronic logic gates could be

reproduced as building blocks for engineered genetic networks then synthetic biologists would have at their disposal a set of well characterised, compatible logic modules which could be used to design more complex systems, and as is the case with circuit design in electronics, simulation and modelling techniques could aid the creation of those designs.

The solutions to the 2-SAT problem were constructed from the bottom-up in a modular fashion, by first creating simple logic gate modules from genes and proteins, and then combining these modules to codify instances of the 2-SAT problem. This modularisation of functionality led to a concise and hierarchical definition of the intended regulatory network design. The P systems representing the designed regulatory networks were distributed in a specific geometrical disposition using a *finite point lattice*. This geometrical distribution gave rise to specific signalling patterns codifying all the possible truth assignments to the corresponding 2-SAT instance. Model checking was then employed to interrogate the models.

8.2 Bacterial Logic Gates

Synthetic biology aims to apply engineering approaches to biological systems and create a set of standardised genetic components with the following properties:

- *Well characterised*: The behaviour of the components should be well understood, and predictable with respect to their response to different inputs.
- *Modular*: Each component should comprise a small, self-contained unit of functionality, with clearly defined inputs and outputs. Components should be composable (e.g. the output of one component can be the input of the other).
- *Orthogonal*: During its operation a component should not interfere with other components, and should not be prone to interference from other components.
- *Robust*: The components should produce output reliably and predictably in a noisy environment and produce the correct output for a wide range of input levels.

In this section the modelling framework introduced in Chapter 3 is used to facilitate the process of designing synthetic gene regulatory networks fulfilling the above criteria and behaving as logic gates for encapsulation within vesicle membranes. The logic gate approach to developing synthetic biology systems was first proposed in [264, 263, 262, 16] and involves the creation of small genetic modules which act as logic gates. These modules can then be interconnected to create more complex genetic circuits. The logic gate abstraction provides a set of components that are inherently modular and well understood.

8.2.1 Transcriptional Logic Gates

Transcriptional regulation is used to create the logic gate modules for vesicle computing. A logic gate is represented by a single gene and its inputs are mRNA encoding transcription factors which bind to the gene promoter and its output is the transcribed mRNA from the corresponding gene. Three different logic gates were modelled in this work, namely, the AND gate, the OR gate and the NOT gate.

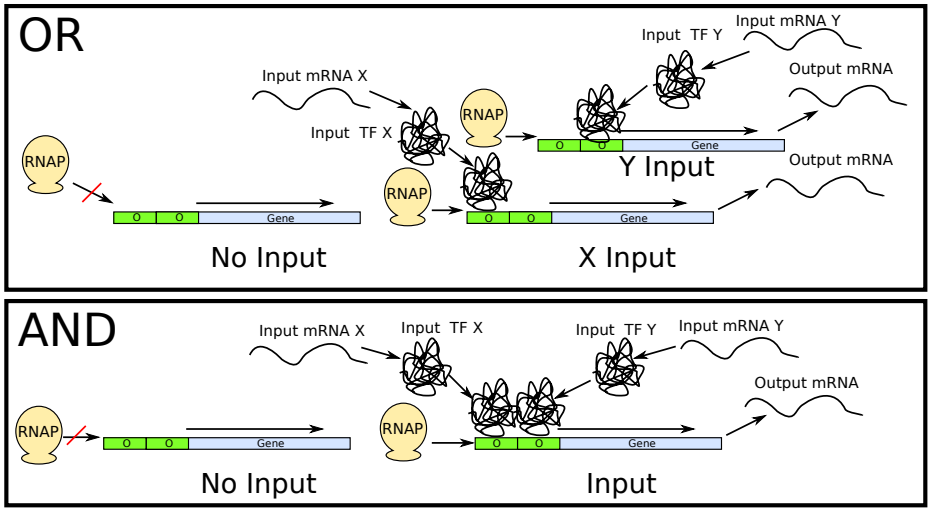


Figure 8.1: The top diagram shows the the operation of the OR gate, in the absence of input mRNA RNA polymerase is unable to bind to the promoter (the section of the gene that is green) and transcription does not occur, the output of the gate is therefore False as no output mRNA is produced. When either of the input activators are transcribed, the input to the gate is considered to be high. The activators bind to the gene operator (labelled O) and allow the RNAP to begin transcription, resulting in the production of output mRNA from the gate meaning that the gate output is high. The bottom diagram illustrates the operation of the AND gate, when no input protein is present the gate does not produce output mRNA and so the output is low. However, when both mRNA inputs are present, the two different activators will bind to the gene promoter and enable transcription to occur, which means that the AND gate produced output mRNA.

The NOT gate follows the same design introduced in Figure 2.5 and is represented by a gene regulated by a repressor, which binds to the gene promoter and prevents transcription if the input is present. Figure 8.1 illustrates the operation of the AND and OR gates. The OR gate is represented by a gene regulated by two different activators, which enable transcription when either or both activators are bound to the promoter. Similarly, the AND gate is regulated by two different activators, but both of then must bind to the promoter before transcription can occur.

Some species of bacteria use quorum sensing [73] to communicate with one another via the production of small signal molecules which diffuse through the bacterial membrane enabling bacteria to coordinate activities as a colony. Recently, these signalling mechanisms have been employed in synthetic biology research as a mechanism for enabling and controlling engineered pathways of gene

expression. A signal transduction module is included in the models and signal molecules are allowed to diffuse through the vesicle membrane and trigger the production of transcription factors which act as the inputs to one or more of the logic gates. In this way, the behaviour of the bacteria can be controlled with the addition or removal of signal molecules from the environment.

8.2.2 Stochastic P System Models of Logic Gates

In what follows, models of the AND, OR and NOT logic gates are presented as stochastic P system modules. The goal is to use these modules as building blocks for the codification of instances of the 2-SAT problem, propositional logic formulas in conjunctive normal form with at most two literals, as synthetic gene regulatory networks. The models assume an underlying metabolism provided by the bacteria or vesicle which supplies the necessary enzymes, nucleotides, amino acids and energy required for transcription and translation of proteins. The algorithm for the evolution of SP and LPP systems requires the association of specific stochastic constants to the rewriting rules forming the different P system modules. In order to keep biological relevance in the models of logic gates characteristic rate constants from gene regulatory systems in *Escherichia coli* are used [14].

Transcription initiation, transcription factor binding and ribosome binding are assumed to occur at the diffusion limited rate of $1min^{-1}$. Transcription factor decomplexation from gene promoters typically occurs at a rate of $1min^{-1}$. For transcription factors that bind cooperatively to a gene promoter already occupied by another transcription factor a decomplexation rate was chosen that was one hundred times slower ($0.01min^{-1}$) in order to model the more stable cooperative binding. The rate at which the RNA polymerase completes transcription is $3.33min^{-1}$ assuming a characteristic gene length of several hundreds of nucleotides and a transcription elongation rate of 40 nucleotides per second. Ribosomes complete translation at a rate of $3.766min^{-1}$ according to a translation elongation of 12 amino acids (the elementary units of proteins) per second. Degradation rates for mRNA and proteins were fixed to $0.139\,min^{-1}$ and $0.023\,min^{-1}$ which correspond to mRNA and protein half lives of 5 and 30 minutes respectively.

In order to produce a modular and reusable design of the logic gate models a module library, *LogicGateLib*, is introduced. The basis of the logic gate modules are a set of modules describing the basic processes of transcription, translation, post-transcriptional regulation, signal sensing and diffusion. As described in Chapter 3, these rule set modules can be composed in stochastic P systems to create complex vesicle computing designs in a modular and hierarchical fashion. Those stochastic P system models can then be distributed over a lattice to form a lattice population P system. Details of each module in the *LogicGateLib* library is given in Tables 8.1 and 8.2, starting with the simplest modules describing the basic regulatory processes.

- *Prep* describes a gene with a promoter where a repressor binds cooperatively. First a repressor protein attaches to the promoter which makes it easier for a second repressor to bind producing a more stable interaction. The gene is only transcribed in the absence of both repressor

Table 8.1: Library of modules *LogiGateLib* used in the design of transcriptional logic gates

Module	Rules	Description
$Prep\left(\begin{array}{c}\{G,R,M\}\\{\{c_1,\ldots,c_5\}}\\{\{l\}}\end{array}\right)$	$[\ G\ ]_l \xrightarrow{c_1} [\ G + M.RNAP\ ]_l$ $[\ R+G\ ]_l \xrightarrow{c_2} [\ R.G\ ]_l$ $[\ R.G\ ]_l \xrightarrow{c_3} [\ R+G\ ]_l$ $[\ R+R.G\ ]_l \xrightarrow{c_2} [\ R.R.G\ ]_l$ $[\ R.R.G\ ]_l \xrightarrow{c_4} [\ R+R.G\ ]_l$ $[\ R.R.G\ ]_l \xrightarrow{c_5} [\ R.G\ ]_l$ $[\ R.G\ ]_l \xrightarrow{c_5} [\ G\ ]_l$	Cooperative gene repression
$Pact\left(\begin{array}{c}\{G,A,M\}\\{\{c_1,\ldots,c_4\}}\\{\{l\}}\end{array}\right)$	$[\ A+G\ ]_l \xrightarrow{c_1} [\ A.G\ ]_l$ $[\ A.G\ ]_l \xrightarrow{c_2} [\ A+G\ ]_l$ $[\ A.G\ ]_l \xrightarrow{c_3} [\ G\ ]_l$ $[\ A.G\ ]_l \xrightarrow{c_4} [\ A.G + M.RNAP\ ]_l$	Activation of a gene G by a single transcription factor A.
$Pact_2\left(\begin{array}{c}\{G,A_1,A_2,M\}\\{\{c_1,\ldots,c_4\}}\\{\{l\}}\end{array}\right)$	$[\ A_1+G\ ]_l \xrightarrow{c_1} [\ A_1.G\ ]_l$ $[\ A_1.G\ ]_l \xrightarrow{c_2} [\ A_1+G\ ]_l$ $[\ A_1.G\ ]_l \xrightarrow{c_3} [\ G\ ]_l$ $[\ A_2+G\ ]_l \xrightarrow{c_1} [\ A_2.G\ ]_l$ $[\ A_2.G\ ]_l \xrightarrow{c_2} [\ A_2+G\ ]_l$ $[\ A_2.G\ ]_l \xrightarrow{c_3} [\ G\ ]_l$ $[\ A_2+A_1.G\ ]_l \xrightarrow{c_1} [\ A_2.A_1.G\ ]_l$ $[\ A_1+A_2.G\ ]_l \xrightarrow{c_1} [\ A_2.A_1.G\ ]_l$ $[\ A_2.A_1.G\ ]_l \xrightarrow{c_2} [\ A_2+A_1.G\ ]_l$ $[\ A_2.A_1.G\ ]_l \xrightarrow{c_2} [\ A_1+A_2.G\ ]_l$ $[\ A_2.A_1.G\ ]_l \xrightarrow{c_3} [\ A_1.G\ ]_l$ $[\ A_2.A_1.G\ ]_l \xrightarrow{c_3} [\ A_2.G\ ]_l$ $[\ A_2.A_1.G\ ]_l \xrightarrow{c_4} [\ A_2.A_1.G + M.RNAP\ ]_l$	Activation of a gene G only when two transcription factors A_1 and A_2 bind to the corresponding promoter.
$P_{OR}\left(\begin{array}{c}\{G,A_1,A_2,M\}\\{\{c_1,\ldots,c_4\}}\\{\{l\}}\end{array}\right)$	$[\ A_1.G\ ]_l \xrightarrow{c_1} [\ A_1.G + M.RNAP\ ]_l$ $[\ A_2.G\ ]_l \xrightarrow{c_1} [\ A_2.G + M.RNAP\ ]_l$ $[\ A_1+G\ ]_l \xrightarrow{c_2} [\ A_1.G\ ]_l$ $[\ A_2+G\ ]_l \xrightarrow{c_2} [\ A_2.G\ ]_l$ $[\ A_1.G\ ]_l \xrightarrow{c_3} [\ A_1+G\ ]_l$ $[\ A_2.G\ ]_l \xrightarrow{c_3} [\ A_2+G\ ]_l$ $[\ A_1.G\ ]_l \xrightarrow{c_4} [\ G\ ]_l$ $[\ A_2.G\ ]_l \xrightarrow{c_4} [\ G\ ]_l$	Activation of a gene G by one of two transcription factors A_1 or A_2.
$PostTransc\left(\begin{array}{c}\{M,P\}\\{\{c_1,\ldots,c_5\}}\\{\{l\}}\end{array}\right)$	$[\ M.RNAP\ ]_l \xrightarrow{c_1} [\ M\ ]_l$ $[\ M\ ]_l \xrightarrow{c_2} [\ ]_l$ $[\ M\ ]_l \xrightarrow{c_3} [\ M + P.Rib\ ]_l$ $[\ P.Rib\ ]_l \xrightarrow{c_4} [\ P\ ]_l$ $[\ P\ ]_l \xrightarrow{c_5} [\ ]_l$	Transcription termination, mRNA degradation, translation initiation, translation termination and protein degradation
$Dim\left(\begin{array}{c}\{A_1,A_2,D\}\\{\{c_1,c_2\}}\\{\{l\}}\end{array}\right)$	$[\ A_1+A_2\ ]_l \xrightarrow{c_1} [\ D\ ]_l$ $[\ D\ ]_l \xrightarrow{c_2} [\ ]_l$	Dimerisation between two proteins and degradation of the dimer.
$Sig\left(\begin{array}{c}\{P,S\}\\{\{c_1,\ldots,c_2\}}\\{\{l\}}\end{array}\right)$	$[\ P\ ]_l \xrightarrow{c_1} [\ P+S\ ]_l$ $[\ S\ ]_l \xrightarrow{c_2} [\ ]_l$	Signal S production by an enzyme P
$Diff\left(\begin{array}{c}\{S\}\\{\{c_1\}}\\{\{l\}}\end{array}\right)$	$[\ S\ ]_l \overset{(0,1)}{\bowtie} [\ ] \xrightarrow{c_1} [\ ]_l \overset{(0,1)}{\bowtie} [\ S\ ]$ $[\ S\ ]_l \overset{(0,-1)}{\bowtie} [\ ] \xrightarrow{c_1} [\ ]_l \overset{(0,-1)}{\bowtie} [\ S\ ]$ $[\ S\ ]_l \overset{(1,0)}{\bowtie} [\ ] \xrightarrow{c_1} [\ ]_l \overset{(1,0)}{\bowtie} [\ S\ ]$ $[\ S\ ]_l \overset{(-1,0)}{\bowtie} [\ ] \xrightarrow{c_1} [\ ]_l \overset{(-1,0)}{\bowtie} [\ S\ ]$	Diffusion of a molecule S in the four possible directions in a rectangular lattice.

proteins. This module takes three inputs, a gene G a repressor R and an mRNA M, the output of the gate. The first rule in this module represents the binding of the RNAP to the gene. RNAP is not represented as an explicit molecular species in our simulations. Instead, the mRNA is tagged with (.RNAP) to indicate that the process of transcription has begun. The second and third rules represent the binding and disassociation of a repressor protein from the gene promoter. The 4th and 5th rules describe the cooperative binding and disassociation of the second repressor protein from the promoter. The last two rules specify the degradation of the the repressor proteins whilst bound to the gene.

- *Pact* represents the interactions whereby a single activator bound to a gene promoter is enough to produce transcription. The first two rules describe the binding and disassociation of the activator A from the gene G. The third rule specifies degradation of the activator whilst bound to the gene promoter. The last rule represents transcription initiation, production of M, only in the situation when the activator is bound to the gene promoter $A.G$.

- $Pact_2$ represents the interactions regulating a gene G that requires two different activators, A_1 and A_2, to bind to its promoter in order to activate transcription, production of $M.RNAP$. The first two rules represent the binding and disassociation of the first activator protein A_1 from the gene promoter. The third rule specifies the degradation of the first activator whilst bound to the promoter. The following three rules describe the same processes for the second activator A_2 protein. The seventh and eighth rules represent the binding of the second activator when one of the activators is already bound. Rules nine to twelve represent the decomplexation and degradation of one activator from the promoter when both activators are bound to it. The last rule represents transcription initiation by RNAP only when both activators ($A_1.A_2.G$) are bound to the gene promoter.

- P_{OR} represents a gene G whose transcription initiation, production of $M.RNAP$, takes place only when either two different activator proteins is bound to its promoter, $A_1.G$ or $A_2.G$, as specified in the first two rules. Rules three to six describe the independent binding and of both activators to the gene promoter. The last two rules model the degradation of the activators whilst bound to the gene promoter.

- $PostTransc$ contains the rules describing transcription termination, translation and degradation processes associated with the protein codified in a specific mRNA. The inputs to the gate are an mRNA M and a protein P. The first rule in the module represents transcription termination of the corresponding mRNA by RNAP. The second rule is the degradation of the mRNA strand. The two following rules represent the binding of a ribosome to the mRNA, translation initiation, and the completion of translation. The last rule specifies the degradation of the protein.

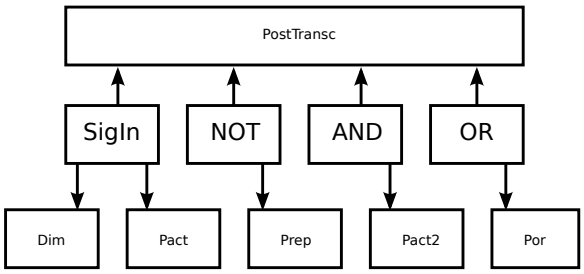


Figure 8.2: A dependency diagram for the TLG modules, modules are shown as boxes, and the arrows between them indicate the module at the arrows source depends on the module at the arrows destination.

- *Dim* represents the formation of a molecular complex D consisting of two proteins, A_1 and A_2, and its subsequent degradation.
- *Sig* represents the synthesis of a signal molecule S by a specific protein (enzyme) P and its degradation.
- $Diff$ describes the diffusion of a signal molecule S in the four different possible directions in a regular rectangular lattice.

In Table 8.2 composite modules are defined to represent transcriptional logic gates (TLGs) as a combination of the modules in Table 8.1. In these new modules the variables to instantiate correspond to the inputs and outputs of the particular TLG. All the stochastic constants associated with the rules are instantiated with characteristic values from E. *coli* as discussed previously and therefore the sets of variables for the stochastic constants are empty. Figure 8.2 shows the dependencies between the TLG modules and the modules from Table 8.1.

- The *NOT* module represents a *NOT logic gate*. The inputs of this TLG are a gene G and the mRNA M_{IN} codifying a repressor protein R. The output is the mRNA M_{NOT} transcribed from the gene G. This module is composed of the $PostTransc$ module, which produces the repressor R from the input mRNA M_{IN} and regulates the different processes of degradation, and the *Prep* module which codifies the molecular interactions responsible for the cooperative repression of the gene G by the repressor R in the production of the output mRNA M_{OUT}.
- The *AND* module specifies an *AND logic gate*. The inputs in this case are two mRNA, M_{A_1} and M_{A_2}, codifying two distinct activators, A_1 and A_2, and a gene G where these two activators can bind. The output as in the rest of the TLGs is the mRNA M_{AND} transcribed from the gene G. The module is composed of two $PostTransc$ modules regulating the post-transcriptional processes associated with both activators and a $Pact_2$ module describing the regulation of the gene G by the two transcription factors A_1 and A_2 leading to the production of the output M_{AND}.

Table 8.2: Extension of the library of modules *LogicGateLib* to include the design of the basic TLGs

Module	**Rules**	**Description**
$NOT\left(\begin{array}{c}\{G, M_{IN}, R, M_{NOT}\}\\ \{\ \}\\ \{l\}\end{array}\right)$	$PostTransc\left(\begin{array}{c}\{M_{IN}, R\}\\ \{3.33, 0.139, 1,\\ 3.766, 0.023\}\\ \{l\})\end{array}\right)$ $Prep\left(\begin{array}{c}\{G, R, M_{NOT}\}\\ \{0.2, 1, 1, 0.01, 0.023\}\\ \{l\}\end{array}\right)$	The NOT module represents a simple NOT gate having as input M_{IN} mRNAs coding for the repressor protein R, and producing M_{NOT} mRNA as output.
$AND\left(\begin{array}{c}\{G, M_{A_1}, M_{A_2},\\ A_1, A_2, M_{AND}\}\\ \{\ \}\\ \{l\}\end{array}\right)$	$PostTransc\left(\begin{array}{c}\{M_{A_1}, A_1\}\\ \{3.33, 0.139, 1,\\ 3.766, 0.023\}\\ \{l\}\end{array}\right)$ $PostTransc\left(\begin{array}{c}\{M_{A_2}, A_2\}\\ \{3.33, 0.139, 1,\\ 3.766, 0.023\}\\ \{l\}\end{array}\right)$ $Pact_2\left(\begin{array}{c}\{G, A_1, A_2, M_{AND}\}\\ \{1, 10, 0.023, 0.5\}\\ \{l\}\end{array}\right)$	The AND module represents a gene activated by two different transcription factors.
$OR\left(\begin{array}{c}\{G, M_{A_1}, M_{A_2},\\ A_1, A_2, M_{OR}\}\\ \{\ \}\\ \{l\}\end{array}\right)$	$PostTransc\left(\begin{array}{c}\{M_{A_1}, A_1\}\\ \{3.33, 0.139, 1,\\ 3.766, 0.023\}\\ \{l\}\end{array}\right)$ $PostTransc\left(\begin{array}{c}\{M_{A_2}, A_2\}\\ \{3.33, 0.139, 1,\\ 3.766, 0.023\}\\ \{l\}\end{array}\right)$ $P_{OR}\left(\begin{array}{c}\{G, A_1, A_2, M_{OR}\}\\ \{1, 1, 1, 0.023\}\\ \{l\})\end{array}\right)$	The OR module represents a gene activated by one of two different transcription factors, which effectively implements an OR gate.
$SigIn\left(\begin{array}{c}\{G_{Rec}, M_{Rec}, Rec,\\ S, A, G_{OUT}, M_{OUT}\}\\ \{\ \}\\ \{l\}\end{array}\right)$	$Prep\left(\begin{array}{c}\{G_{Rec}, rep, M_{Rec}\}\\ \{0.2, 1, 10, 0.1, 0.069\}\\ \{l\}\end{array}\right)$ $PostTransc\left(\begin{array}{c}\{M_{Rec}, Rec\}\\ \{3.33, 0.139, 1,\\ 3.766, 0.069\}\\ \{l\}\end{array}\right)$ $Dim\left(\begin{array}{c}\{Rec, S, A\}\\ \{1, 0.19\}\\ \{l\}\end{array}\right)$ $Pact\left(\begin{array}{c}\{G, A, M_{OUT}\}\\ \{1, 0.01, 0.19, 5\}\\ \{l\}\end{array}\right)$	The $SigIn$ module represents the activation of a gene G_{OUT} when a signal is sensed by a receptor protein Rec codified in the gene G_{Rec}.

- The *OR* module implements an *OR logic gate*. The inputs here are two mRNA, M_{A_1} and M_{A_2}, codifying two activators, A_1 and A_2, and a gene G that can be activated by either of the activators. The output is the mRNA M_{OR} transcribed from the gene G. The module is composed of two $PostTransc$ modules describing the post-transcriptional processes of both activators and a P_{OR} module describing the regulation of the gene G by the two transcription factors A_1 and A_2 in the production of the output M_{OR}.
- The $SigIn$ module consists of a transduction system that converts a small diffusible signal into the mRNA acting as the inputs to the TLGs. The module is composed of a $Prep$ module and a $PostTransc$ modules responsible for the production of a receptor protein Rec. The signal S interacts with this receptor according to the module Dim to produce the activator A. Finally, this activator enables the transcription of the output mRNA as specified in the module $Pact$.

Using these modules a codification of the instances of the 2-SAT problem was developed from synthetic gene regulatory networks and a codification of the possible truth assignments using spatial signal patterns as discussed in the next two sections. Stochastic model checking was used to verify some properties of the dynamics of the proposed codifications.

8.3 A solution to the 2-SAT problem

Modules from the library *LogicGateLib* were used to construct a solution to a subset of the instances of the Boolean 2-satisfiability problem, 2-SAT, with one or two variables. The definition of the 2-SAT problem is given below.

Definition 1 (2-SAT problem)**.** *Given a set of Boolean variables x_i where $1 \leq i \leq n$, an instance of the 2-SAT problem F, is defined as a conjunction of clauses $F = c_1 \wedge, ..., \wedge c_k$ where each clause is a disjunction of one or two literals $a_l \in \{x_i, \neg x_i\}$, the satisfiability problem (2-SAT) involves finding an assignment of values for the x_i variables such that F is satisfied. If no such solution exists, then F is said to be unsatisfiable.*

The 2-SAT problem can be solved in linear time by generating a graph of implications and finding the *strong components* of the graph. In terms of space complexity [20] the 2-SAT problem is NL-complete (see Chapter 2). This is one of the "hardest" problems which can be solved with a non-deterministic Turing machine using only a logarithmic amount of memory [207]. In this work, only instances of the 2-SAT problem where $n = 2$ are considered, and so the number of different clauses which can be expressed is 6.

8.3.1 A Spatial Codification of Truth Assignments

The codification of the truth assignment for a Boolean variable x_i in the solution to the 2-SAT problem is achieved by assigning to each variable a signal molecule s_i. If a signal molecule is present within a cell volume, then the corresponding Boolean variable is True, (and is False otherwise).

To illustrate the functionality of the logic gates, P systems containing them were placed in a lattice. Two different signals representing the different inputs to the gates were produced at the side of the lattice by layers of signal producing cells. Signal diffusion between different cells on the lattice was modelled by including the Sig module in each SP system definition. As the signal molecules will degrade at a certain rate, a concentration gradient will form across the lattice for each signal. The cells in close proximity to the senders will contain a high concentration of signal molecules. As the distance from the sender cells increases, the concentration in the cells will decrease until at a distance of roughly half the lattice, where the cells will no longer contain any signal as it will have all degraded before reaching them. The signal sending cells are specified by the SP systems $\mathcal{SP}_{s1}$ and $\mathcal{SP}_{s2}$ in Table 8.3.

Figure 8.3 shows a schematic layout of the lattice, and illustrates how different regions of the lattice correspond to different truth assignments. The boundary cells in the figure are specified with the following stochastic P system.

$$\mathcal{SP}_{Boundary} = (\{\}, [], \{cell\}, \{\}, \{\}) \tag{8.1}$$

When a signal molecule diffuses into a boundary compartment, the signal molecule will remain there as there are no rules which enable the signal to pass through the boundary cell membrane. The boundary cells prevent the accumulation of signal molecules in the cells at the edge of the lattice, which would introduce artifacts into the results of the simulation.

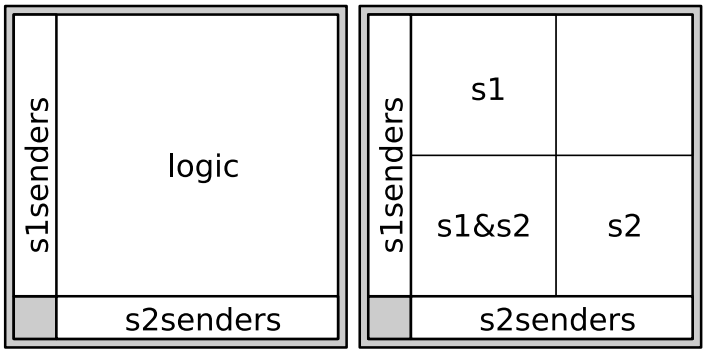


Figure 8.3: The left image shows a schematic representation of the lattice, the dark grey regions denote the boundary cells, and the areas labelled "s1senders" and "s2senders" show the signal sending cells for signal $s1$ and $s2$. The large square region labelled "logic" indicates the regions where the cells containing the logic gates are placed. The right hand image shows the spatial codification of truth assignments variables, based on the diffusion of the corresponding signal molecules across the lattice, cells close to the s1/s2 signal sending cells contain a high concentration of signal, which corresponds to x1/x2 being True. Outside of the area of diffusion for a signal the cells contain no signal molecules, which corresponds to a value of False for the variable.

A signal transduction module $SigIn$ was placed inside each P system so that the presence of signal molecules triggered the production of the activator/repressor inputs to the gates. A simulation

Table 8.3: The SP systems created for the logic gate experiments, the $\mathcal{SP}_{s1}$ and $\mathcal{SP}_{s2}$ systems are responsible for the production of $s1$ and $s2$, the signal molecules which correspond to the x_1 and x_2 truth values respectively.

$$\mathcal{SP}_{s1} = (M_{s1}, \mu_{s1}, L_{s1}, I_b, R_b) \quad \text{where}$$
$$M_{s1} = \{gS1, reg.gS1, reg.reg.gS1, rI1, I1, I2, s1, s2, reg\}$$
$$\mu_{s1} = [\ ]$$
$$L_{s1} = \{b\}$$
$$I_b = \{gene\}$$
$$R_b = \begin{cases} Prep(\{gS1, reg, rI1\}, \{2, 1, 10, 0.1, 0.181\}, \{b\}) \cup \\ PostTransc(\{rI1, I1\}, \{3.33, 0.181, 1, 3.766, 0.181\}, \{b\}) \cup \\ Sig(\{I1, s1\}, \{3.62, 0.181\}, \{b\}) \cup \\ Sig(\{I2, s2\}, \{3.62, 0.181\}, \{b\}) \end{cases}$$

$$\mathcal{SP}_{s2} = (M_{s2}, \mu_{s2}, L_{s2}, I_b, R_b) \quad \text{where}$$
$$M_{s2} = \{gS2, reg.gS2, reg.reg.gS2, rI1, I1, I2, s1, s2, reg\}$$
$$\mu_{s2} = [\ ]$$
$$L_{s2} = \{b\}$$
$$I_b = \{gene\}$$
$$R_b = \begin{cases} Prep(\{gS2, reg, rI2\}, \{2, 1, 10, 0.1, 0.181\}, \{b\}) \cup \\ PostTransc(\{rI2, I2\}, \{3.33, 0.181, 1, 3.766, 0.181\}, \{b\}) \cup \\ Sig(\{I1, s1\}, \{3.62, 0.181\}, \{b\}) \cup \\ Sig(\{I2, s2\}, \{3.62, 0.181\}, \{b\}) \end{cases}$$

$$\mathcal{SP}_{NOT} = (M_N, \mu_N, L_N, I_b, R_b) \quad \text{where}$$
$$M_N = \{G_{R1}, M_{R1}, R1, s1, A_{s1}, G_{s1}, M_{s1}, I1, G_N, M_{IN}, R, M_N, P_N\}$$
$$\mu_N = [\ ]$$
$$L_N = \{b\}$$
$$I_b = \{gene\}$$
$$R_b = \begin{cases} SigIn(\{G_{R1}, M_{R1}, R1, s1, A_{s1}, G_{s1}, M_{s1}\}, \{\}, \{b\}) \cup \\ Sig(\{I1, s1\}, \{3.62, 0.181\}, \{b\}) \cup \\ NOT(\{G_N, M_{s1}, R_N, M_N\}, \{\}, \{b\}) \cup \\ PostTransc(\{M_N, P_N\}, \{3.33, 0.139, 1, 3.766, 0.023\}, \{\}, \{b\}) \end{cases}$$

$$\mathcal{SP}_{AND} = (M_A, \mu_A, L_A, I_b, R_b) \quad \text{where}$$
$$M_A = \{G_{R1}, M_{R1}, R1, s1, A_{s1}, G_{s1}, M_{s1}, G_{R2}, M_{R2}, R2, s2, A_{s2}, G_{s2}, M_{s2}, I1, I2, G_A, A_{s1}, A_{s2}, M_A, P_A\}$$
$$\mu_A = [\ ]$$
$$L_A = \{b\}$$
$$I_b = \{gene\}$$
$$R_b = \begin{cases} SigIn(\{G_{R1}, M_{R1}, R1, s1, A_{s1}, G_{s1}, M_{s1}\}, \{\}, \{b\}) \cup \\ SigIn(\{G_{R2}, M_{R2}, R2, s2, A_{s2}, G_{s2}, M_{s2}\}, \{\}, \{b\}) \cup \\ Sig(\{I1, s1\}, \{3.62, 0.181\}, \{b\}) \cup \\ Sig(\{I2, s2\}, \{3.62, 0.181\}, \{b\}) \cup \\ AND(\{G_A, M_{s1}, M_{s2}, A_{s1}, A_{s2}, M_A\}, \{\}, \{b\}) \cup \\ PostTransc(\{M_A, P_A\}, \{3.33, 0.139, 1, 3.766, 0.023\}, \{\}, \{b\}) \end{cases}$$

$$\mathcal{SP}_{OR} = (M_O, \mu_O, L_O, I_b, R_b) \quad \text{where}$$
$$M_O = \{G_{R1}, M_{R1}, R1, s1, A_{s1}, G_{s1}, M_{s1}, G_{R2}, M_{R2}, R2, s2, A_{s2}, G_{s2}, M_{s2}, I1, I2, G_O, A_{s1}, A_{s2}, M_O, P_O\}$$
$$\mu_O = [\ ]$$
$$L_O = \{b\}$$
$$I_b = \{gene\}$$
$$R_b = \begin{cases} SigIn(\{G_{R1}, M_{R1}, R1, s1, A_{s1}, G_{s1}, M_{s1}\}, \{\}, \{b\}) \cup \\ SigIn(\{G_{R2}, M_{R2}, R2, s2, A_{s2}, G_{s2}, M_{s2}\}, \{\}, \{b\}) \cup \\ Sig(\{I1, s1\}, \{3.62, 0.181\}, \{b\}) \cup \\ Sig(\{I2, s2\}, \{3.62, 0.181\}, \{b\}) \cup \\ OR(\{G_O, M_{s1}, M_{s2}, A_{s1}, A_{s2}, M_O\}, \{\}, \{b\}) \cup \\ PostTransc(\{M_O, P_O\}, \{3.33, 0.139, 1, 3.766, 0.023\}, \{\}, \{b\}) \end{cases}$$

was performed for 720 minutes of simulated time, and the levels of the output protein from each gate were plotted as a surface map and Figure 8.4 shows the results of these simulations. Cells containing the logic gate modules (those cells within the "logic" region in Figure 8.3) are producing the correct output. The NOT gate cells which are within the diffusion radius of the input signal produce no output, whereas those outside of the diffusion radius expressed the output protein. For the AND gate the output protein is expressed only in the region where both signals are present, and for the OR gate protein was expressed in those regions where either of the two signals is present.

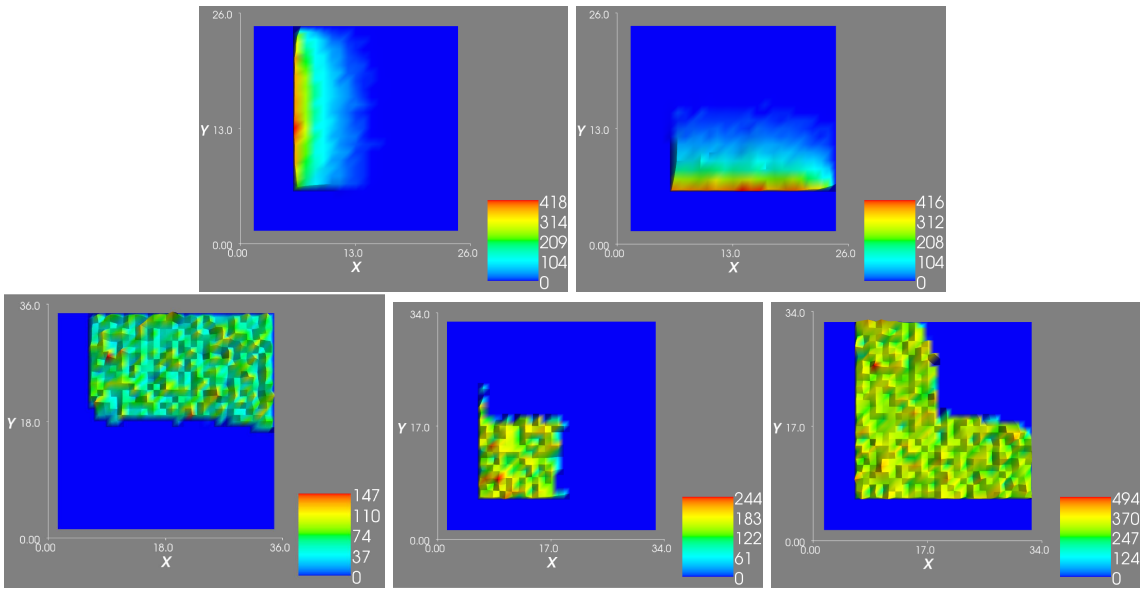


Figure 8.4: Surface map plots of the results of the logic gate simulations. The images on the top row show surface maps for the levels of input signal. The top left and top right images show surface maps for the signals $s1$ and $s2$ diffusing from signal sender cells at the left and bottom edges of the lattice respectively. The bottom row shows surface maps of the expression levels of the output proteins for the NOT gate (bottom left), AND gate (bottom centre) and the OR gate (bottom right).

8.3.2 A Codification of 2-SAT Instances

The codification of the instances of the 2-SAT problem is now described. Given an instance of the 2 variable 2-SAT problem composed of l literals $F = (a_1 \vee a_2) \wedge, ..., \wedge (a_{l-1} \vee a_l)$ for each clause $(a_i \vee a_j)$ $a_i, a_j \in \{x_1, \neg x_1, x_2, \neg x_2\}$, the codification of the four different types of clauses into a collection of logic gate modules is defined as *clauseRules*. The codification requires the label of a compartment

m where the corresponding logic gate modules will be placed.

$$\begin{aligned}
clauseRules((x_i \vee x_j)_k, m) &\rightarrow OR(\{g_k, m_i, m_j, A_i, A_j, m_k\}, \{\}, \{m\}) \\
clauseRules((\neg x_i \vee x_j)_k, m) &\rightarrow NOT(\{g_k N_i, m_i, a_i, m_k N_i\}, \{\}, \{m\}) \cup \\
&\quad OR(\{g_k, m_k N_i, M_j, a_k N_i, a_j, m_k\}, \{\}, \{m\}) \\
clauseRules((\neg x_i \vee \neg x_j)_k, m) &\rightarrow NOT(\{g_k N_i, m_i, a_i, m_k N_i\}, \{\}, \{m\}) \cup \\
&\quad NOT(\{g_k N_j, m_j, a_j, m_k N_j\}, \{\}, \{m\}) \cup \\
&\quad OR(\{g_k, m_k N_i, m_k N_j, a_k N_i, a_k N_j, m_k\}, \{\}, \{m\} \quad (8.2)
\end{aligned}$$

The codification *allClauseRules* can then be defined which generates the necessary modules for all clauses in a problem instance.

$$allClauseRules(c_1 \wedge, ..., \wedge c_k, m) = \bigcup_{a=1}^{k} clauseRules(c_a, m) \quad (8.3)$$

The rules representing the conjunction between the clauses are included in the P system by instantiating the following modules. For a compartment labelled m, and a problem instance F the function *conj* generates these modules.

$$\begin{aligned}
conj(F, m) = \bigcup_{a=1}^{k-1} AND(\quad \{ \quad & G_{CaCa+1AND}, M_{CaOR}, M_{Ca+1OR}, \\
& P_{CaOR}, P_{Ca+1OR}, M_{CaCa+1AND}\}, \{\}, \{m\}) \quad (8.4)
\end{aligned}$$

Given an instance of the 2-SAT problem, F, the SP system $\mathcal{SP}_{2SAT}$ representing it using the above codifications is presented next.

$\mathcal{SP}_{2SAT} = (M_N, \mu_N, L_N, I_b, R_b)$ where

$M_N = \{g_k, m_i, m_j, A_i, A_j, m_k, g_k N_i, a_i, m_k N_i, G_{R1}, M_{R1}, R1, s1, A_{s1}, G_{s1},$
$M_{s1}, G_{R2}, M_{R2}, R2, s2, A_{s2}, G_{s2}, M_{s2}, G_{CaCa+1AND}, M_{CaOR}, M_{Ca+1OR},$
$P_{CaOR}, P_{Ca+1OR}, M_{CaCa+1AND}\}$

$\mu_N = [\ ]$

$L_N = \{cell\}$

$I_b = \{g_k^1, g_k N_i^1, G_{R1}^1, G_{s1}^1, G_{R2}^1, G_{s2}^1\}$

$$R_b = \begin{cases} conj(F, cell) \cup \\ allClauseRules(F, cell) \cup \\ SigIn(\{G_{R1}, M_{R1}, R1, s1, A_{s1}, G_{s1}, M_{s1}\}, \{\}, \{cell\}) \cup \\ SigIn(\{G_{R2}, M_{R2}, R2, s2, A_{s2}, G_{s2}, M_{s2}\}, \{\}, \{cell\}) \end{cases}$$

To model instances of the 2-SAT problem, a lattice of 37×37 cells is created, according to Definition 3.3.1.

$$Lat_{2SAT} = (0, 36, 0, 36) \quad (8.5)$$

The lattice contains boundary cells on the edges of the lattice, SP_{s1} cells along the left edge of the lattice, and $\mathcal{SP}_{s2}$ cells along the bottom edge of the lattice. The rest of the cells in the lattice are 2-SAT cells. The function Pos_{2SAT} maps positions on the lattice to the SP systems $(\mathcal{SP}_{Boundary}, \mathcal{SP}_{s1}, \mathcal{SP}_{s2}, \mathcal{SP}_{2SAT})$ and is defined as follows.

$$\begin{aligned} Pos_{2SAT} &= ((i,j), Boundary)\ i = 0, \ldots, 36\ j = 0, 36 \\ &\quad ((i,j), Boundary)\ i = 0, 36\ j = 1, \ldots, 36 \\ &\quad ((i,j), Boundary)\ i = 1, \ldots, 5\ j = 1, \ldots, 5 \\ &\quad ((i,j), s1)\ i = 6, \ldots, 35\ j = 1, \ldots, 5 \\ &\quad ((i,j), s2)\ i = 1, \ldots, 5\ j = 6, \ldots, 35 \\ &\quad ((i,j), 2SAT)\ i = 6, \ldots, 35\ j = 6, \ldots, 35 \end{aligned} \tag{8.6}$$

The diffusion of the signal molecule between cells in the lattice was modelled by instantiating the *Sig* module for each SP system except for the boundary SP system. The translocation rules for the 2SAT LPP system can then be defined.

$$\begin{aligned} DiffRules &= Diff(\{s1\}, \{1\}, \{m\}) \cup Diff(\{s2\}, \{1\}, \{m\}) \\ TransLoc &= (\emptyset, DiffRules, DiffRules, DiffRules) \end{aligned} \tag{8.7}$$

The LPP system for solving the 2 variable instance of the 2-SAT problem is defined as follows

$$\mathcal{LPP}_{2SAT} = (Lat_{2SAT}, (\mathcal{SP}_{Boundary}, \mathcal{SP}_{s1}, \mathcal{SP}_{s2}, \mathcal{SP}_{2SAT}), Pos_{2SAT}, TransLoc) \tag{8.8}$$

8.4 Results and Discussions

To illustrate the behaviour of the lattice population P systems, two instances of the two variable 2-SAT problem were selected, one which is satisfiable (Eq. 8.9), and one which is not satisfiable (Eq. 8.10).

$$(x_1 \vee \neg x_2) \wedge (x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2) \tag{8.9}$$

$$(x_1 \vee x_2) \wedge (\neg x_1 \vee x_2) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_2) \tag{8.10}$$

Lattice populations P systems were constructed as described above, with the cells producing the s_1 signal placed along the left edge of the lattice, such that they generate a signal gradient across the lattice from the left to the right. The cells producing the s_2 signal are placed along the bottom edge of the lattice, which generates a signal gradient from the bottom of the lattice upwards. As the signals molecules are subject to degradation and diffusion of signal between cells is not instantaneous, a gradient of signal forms across the lattice. The model instances were simulated once

using the Multigillespie Algorithm with Queue (Algorithm 2) for a simulated time of 2880 minutes and data was analysed and plotted using the infobiotics workbench (`http://www.infobiotic.org/infobiotics-workbench/`).

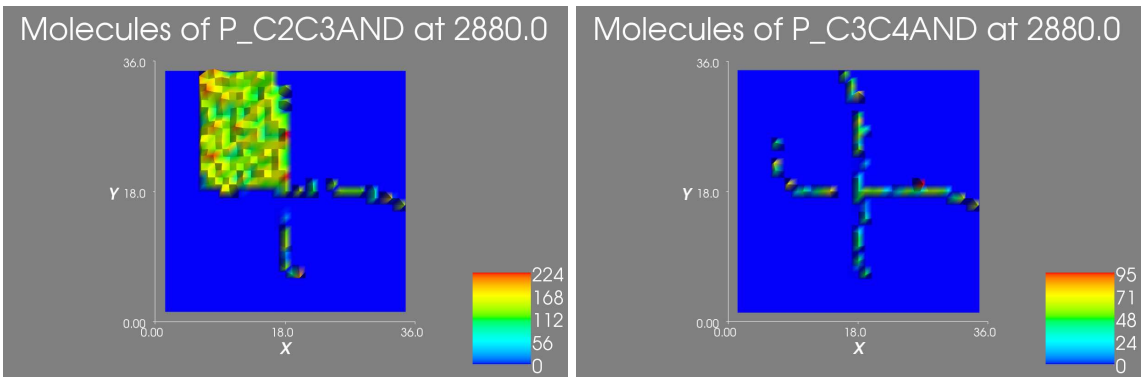


Figure 8.5: Results of the simulation of spatial 2SAT LPP system. The image on the left shows the surface plot for the simulation of the formula given in Equation 8.9, which shows that the formula is satisfiable when x_1 is high and x_2 is low. The image on the right shows the surface plot for the simulation of the formula given in Equation 8.10, which is not satisfiable, and so very little output protein is expressed.

The output from simulating the satisfiable instance of the 2-SAT problem (eq. 8.9) is shown in the left hand image of Figure 8.5. The output of the 2SAT solver cells (protein $P\_C3C4AND$) was fully expressed in the top left region of the lattice, indicating that the instance is satisfiable when x_1 is True and x_2 is false. Note that in both of the results described in Figure 8.5 there were anomalous outputs from the $\mathcal{SP}_{2SAT}$ cells, where a number of cells expressed the output protein incorrectly along the border between the regions where the signal is present and not present. To understand why these errors occur, the anomalous regions in the Eq. 8.9 instance are considered in more detail. First, a cell is chosen for each of the cases where the cell is producing erroneous output. For the cells where the arc of erroneous expression is roughly parallel with the x axis, cells at positions $(25, 18)$, $(25, 20)$ and $(25, 15)$ are considered, the first of which is producing erroneous output. The cells at the second and third position are in areas where the x_2 signal is low and high respectively and provide a comparison of the expression levels between working and non-working cells. The 3-clause instance of the 2-SAT problem which is being solved has the following structure

$$(x_1 \vee \neg x_2) \wedge (x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2) \tag{8.11}$$

The cell at region $(25, 20)$ on the lattice is far enough away from the s_1 and s_2 signal senders so no signal diffuses into the cell. This means that x_1 and x_2 variables in the above instance are both false. The truth values for the clauses in the formula become:

$$True \wedge False \wedge True = False \tag{8.12}$$

Figure 8.6 shows the protein expression levels for each clause in the model of the instance Eq.8.9 for the SP systems at positions $(25, 15)$, $(25, 18)$ and $(25, 20)$. At positions $(25, 15)$ and $(25, 20)$ the output of one of the clauses was clearly false, and so the output of the AND gate modules which link the clauses together was also false. In the case of position $(25, 18)$, two of the clauses fluctuate between False and True values, resulting in the incorrect expression of the output protein. The SP

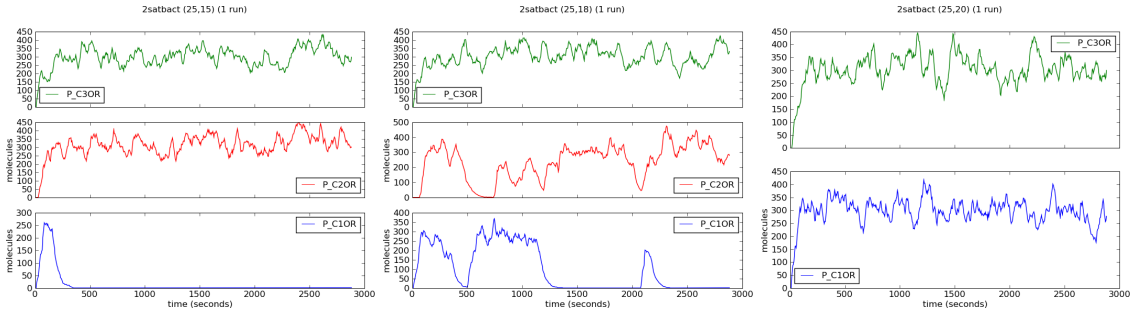


Figure 8.6: The expression levels for the output protein for SP systems at positions $(25, 15)$ left, $(25, 18)$ centre and $(25, 20)$ right. The SP system at position $(25, 15)$ is within the diffusion range of the signal representing the x_2 variable, which assigns it a value of True for that SP system, which produces the correct output for its truth assignment. Likewise, the SP system at position $(25, 20)$, is outside of the range of x_2 signal diffusion and so the x_2 has the False value assigned to it in this SP and therefore produces the correct result. The SP system at $(25, 18)$ is at the boundary of the x_2 signal diffusion range, and so its truth value for this variable is not well defined, leading to fluctuations in the outputs of clauses 1 and 2 due to transient pulses of signal.

system at position $(25, 15)$ in the lattice is outside of the signal diffusion range for the s_1 signal, but should be within the signal diffusion range for the s_2 signal. The x_1 variable in the 3 clause 2-SAT expression is then False, and the x_2 variable is True, resulting in the following truth values for each of the clauses.

$$False \wedge True \wedge True = False \tag{8.13}$$

The above results are now compared with those from the cell at position $(25, 18)$ on the lattice, which is an example of a cell which is producing incorrect output. The cell is at the very edge of the area in which the x_2 signal diffuses, and so x_1 is False, but x_2 is unknown. The truth value of the 3-clause 2-SAT instance can then be reduced to:

$$(False \vee \neg x_2) \wedge (False \vee x_2) \wedge (True \vee \neg x_2) \tag{8.14}$$

what can be further reduced to:

$$\neg x_2 \wedge x_2 = False \tag{8.15}$$

As the cell is at the very edge of the area in which x_2 signal diffuses, it is likely that x_2 will be false for most of the simulation, but transient pulses of signal will occur due to the stochasticity of the environment, and so occasionally x_2 may become true. Regardless of these transient pulses,

the formula should always evaluate to False, as a conjunction of a variable and its negation is always False.

The third clause is outputting True, which is correct. However, the first and the second clauses are fluctuating between True and False. The first clause should be false when x_2 is true, and the second clause should be true when x_2 is true. Figure 8.7 shows an overlay of the output of clauses two and four. The figure shows that, although the levels of P_{C1OR} drop when the levels of P_{C2OR} rise, there is a significant lag time between the change of one output and the change of another, meaning that while one output is rising and the other is falling, both will be considered True until the output which is dropping drops below the level which activates the AND gate.

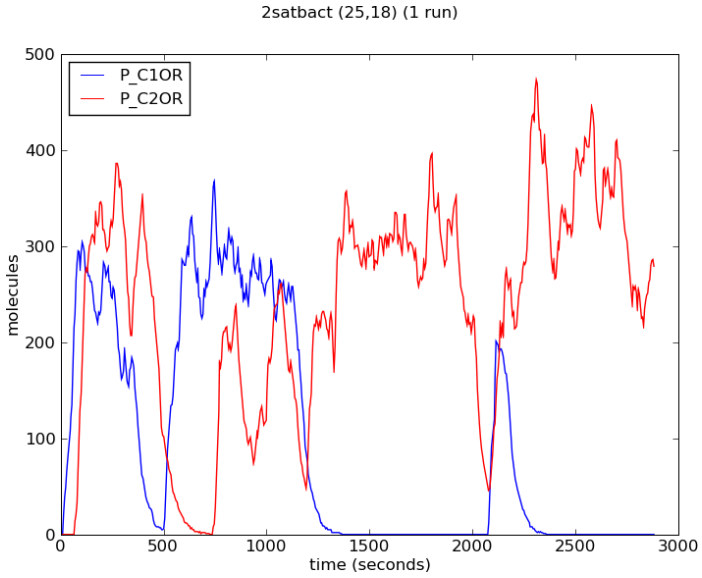


Figure 8.7: Overlay of the protein expression levels for the output of the first and second clause of the SP at position $25, 18$. For several periods during the simulation, the levels of both of the output proteins are high, corresponding to a True value, despite the fact that the logical expressions they represent are contradictory.

Model checking techniques are applied to the logic gate modules described above to investigate this effect further. A simplified model was created with two OR gate modules, with the initial multiset of the P system configured so that one of the OR gates was in a True state (e.g. the number of its output proteins and mRNA were set high initially) but the gate was no longer activated, so that the amount of output protein and mRNA would decrease in the same way as they would in the 2SAT model. The other OR gate was initially set so that its output was low, but an input to the gate was present, so that the output would rise.

The following reward structure was added to the model, so that reward would accumulate when the number of output proteins was greater than the threshold, the parameter that was varied during the model checking. The property of the model which was being investigated was the length of time that both OR gate outputs were higher than the *threshold* parameter, which was varied

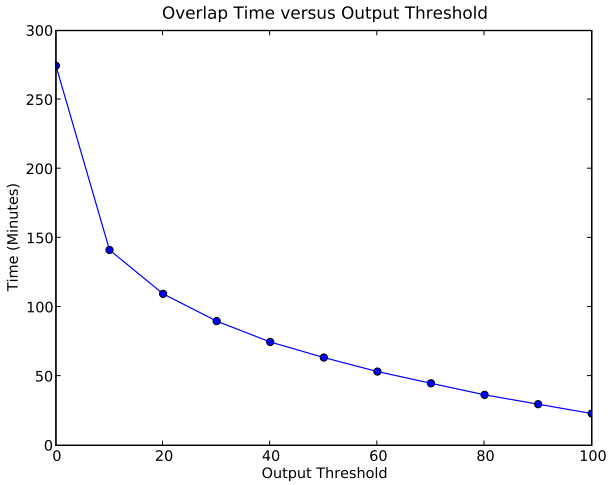


Figure 8.8: The period of time over which both output proteins are present for a given output threshold level

between 0 and 100 in steps of 10. This property is investigated with the following query in continuous stochastic logic:

```
rewards
    P_OR1 > threshold & P_OR2 > threshold : 1;        R =?[C <= 500]
endrewards
```

1000 runs were performed for each parameter, with $\epsilon = 0.08$ and $\delta = 0.05$, Figure 8.8 shows a plot of the results of this analysis.

For an output threshold of 10 proteins, which is typically enough to activate/repress all of the model logic gates, both proteins were high for nearly 150 minutes. The transcription and degradation rates of the OR gate output proteins are the cause of the overlap, as if the transcription is faster than the degradation, then an overlap will occur. In order to determine how the degradation and transcription rate parameters affected the probability of overlap occurring, model checking of the same model was performed with the following query in continuous stochastic logic, which determines the probability of an overlap occurring. The transcription and degradation rates were altered from 0.01 to 1.01 in increments of 0.1. Figure 8.8 shows the result of this analysis.

```
P =? [ true U[0,500] P_OR1 > 10 & P_OR2 > 10]
```

The figure indicates that a transcription rate of 0.2, with a degradation rate of 0.6, should reduce the probability of overlap occurring to close to zero. The parameters for this instance of the 2-SAT model were then altered to these values, and the simulation repeated. The results of the simulation are shown in Figure 8.10.

Resimulating the 3-clause instance of the problem with altered parameters based on model checking shows that the number of cells producing incorrect output is greatly reduced. However, a

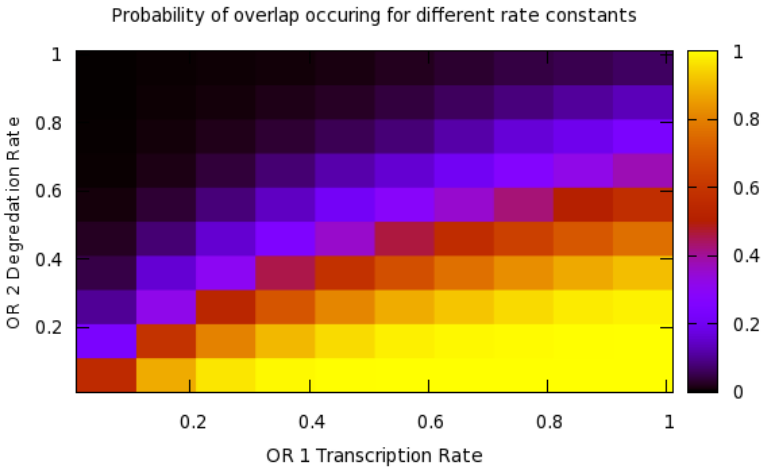


Figure 8.9: The relationship between the OR1 transcription rate and the OR2 degradation rate, and the probability of an overlap between the OR1 and OR2 outputs occurring.

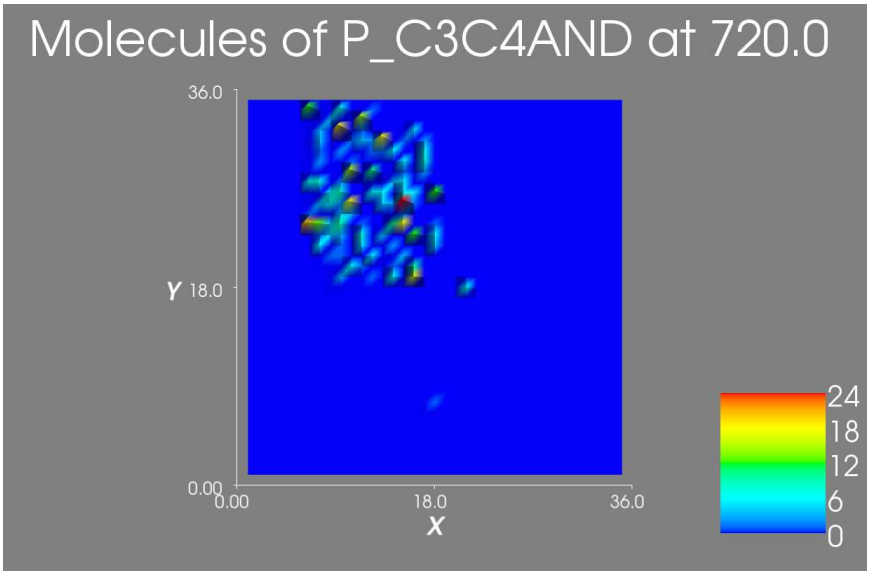


Figure 8.10: The result of simulating the 4-clause instance of the 2-SAT problem with altered transcription and degradation parameters for the OR gate output proteins.

further consequence of altering the rates is that the equilibrium expression levels of the cells which are active and outputting correctly has been reduced by around a factor of 10.

8.5 Conclusions

In this chapter a modular approach to the design of a population based vesicle or cellular computing systems based on stochastic population P systems, using the logic gate abstraction to create a solution to a subset of the instances of the 2-SAT problem has been described. Although the 2-SAT problem is not a "hard" problem in terms of computational complexity, it provides a useful example for demonstrating the use of lattice population P systems, in combination with model checking, as a new modelling technique inspired by the concepts of executable biology.

8.5.1 Comparison with Electronic Logic Gates

Despite the ease with which the logic gate abstraction can be expressed using gene transcriptional regulation there are some key differences between the operating environment and behaviour of electronic and transcriptional logic gates (TLG). These differences are now considered in detail.

- *Stochasticity*: The inputs to the TLGs are typically of the order of tens or hundreds of proteins, and so the signal to noise ratio may be quite low in comparison to electronic logic gates. Moreover, the level of noise may be altered by factors external to the cell such as temperature.

- *Interference*: In electronics, components can suffer from electromagnetic interference when placed in close proximity to one another without shielding. The risk of interference between TLGs is much greater, as the signals and outputs of the gates diffuse freely throughout the same medium, reducing the degree of modularity and orthogonality of a system. In order to determine if TLGs interfere with one another, the designer must check and compare the components of the systems. Also, there is the possibility of unintended interactions between TLGs and the underlying biomachinary of the living cell which is more difficult to control and prevent.

- *Limited number of gates*: The transcription factors which act as inputs to the TLGs must be different from one another in order to ensure that no unintended interference between logic gates occurs. This also implies that the promoter regions for each input for the gates must be different. Therefore, the number of required distinct transcription factors and promoters increases linearly with the complexity of the logic circuit built using TLGs. Compartmentalising TLGs within vesicles to form modules with clearly defined inputs and outputs may help to reduce the number of different trascription factors required to express a circuit. Nevertheless, the number of well characterised promoters and transcription factors in bacterial systems is limited. There is an additional limit imposed by the capability of the living cells to satisfy the metabolic load, demand for energy, necessary in order to sustain the introduced number of logic gates.

- *Large propagation delay*: The propagation delay of a logic gate is defined as the time required for the output logic state of the gate to change after the input is altered. In electronic logic gates this delay is typically nanoseconds in length whereas in TLGs the delay can range from minutes to hours depending on the rate at which the output proteins degrade and the input transcription factors bind to the promoters.

Despite these differences, logic gates are still a useful abstraction when designing synthetic biology systems. Although the focus has been on transcriptional gene regulation, biology provides a number of different gene regulatory mechanisms which could be employed, such as RNA interference. Other features of cell biology such as the membrane may be used to support modularity.

The solutions to the problem 2-SAT described in this chapter were constructed by creating logic gates based on transcriptional regulation by transcription factors, from the bottom up. 2-SAT instances were encoded within model vesicles or bacteria in a spatial lattice, and depending on the position of the vesicles on the lattice, the vesicles were subject to different combinations of inputs.

The 2-SAT models functioned correctly, and the majority of the 2-SAT SP systems on the lattice produced the correct output. However, a small number of cells at the boundaries of the regions of signal diffusion were being erroneously expressed. Model checking revealed that this was due to the relatively slow degradation rates of the proteins in comparison with transcription rates, and investigation of these parameters suggested an alternative degradation and transcription rate which greatly reduced the amount of erroneous expression. Discovering these problems during simulation illustrates two key aspects of the work relevant to design in synthetic biology. Firstly, that simulation and modelling, combined with the investigative power of model checking provides the synthetic biologist with a means to anticipate and understand likely pitfalls and subtle errors in a design before committing to an *in vivo* implementation. Secondly, although the mapping of logic gates onto biological systems is a useful abstraction, the resulting transcriptional logic gates as described in this work cannot be considered as "black box" modules, due to the fundamental differences in environment and process of operation.

For each new instance of the 2-SAT problem, a model vesicle must be created which contains the necessary TLGs to solve the instance, which would be a very time consuming and inefficient method of solving such simple problems, and so the solution presented here is not a general one. However, the instances of the 2-SAT problem and the described solutions provide a good pedagogical example of how the methodology can be used to design and evaluate solutions in synthetic biology.

Overall, the approach to designing computational models in biology has a number of advantages over other techniques. At the level of the individual stochastic P systems the modular approach to rule specification enabled the modeller to design the system in a "bottom-up" fashion, combining reusable modules which contain smaller elements of functionality into a stochastic P system representing a solution to a given problem. The technique also provides modularity at a higher level of abstraction, in that a stochastic P system can act as a template, defining the functionality

of many individual cells in a lattice structure. The lattice gives each cell a location in space, which enables simulations of populations of P systems and the visualisation of gene expression in space and time. This is important as currently synthetic biology is mostly practised on micro-organisms, where the stochastic nature of the cell environment means that the property being observed may vary greatly across different cells, and so lattice population P systems allow an ensemble analysis of large numbers of simulated entities, which should more closely resemble the distribution of values found in a colony of vesicle or bacteria. The addition of spatial properties to the cells in the lattice enables simulation of synthetic biology designs for larger numbers of cells in the presence of concentration gradients, and experimentation with inter-cellular signalling designs.

Chapter 9

Discussion and Conclusions

9.1 Overview and Contributions

This thesis has attempted to address the research questions posed in Chapter 1, which are as follows:

- How might a simple chemical computational device, encapsulated within a vesicle be designed?
- How can computer science aid the development of vesicle computing with simulation and modelling techniques?

In attempting to answer the first question, this thesis considered the possibility of utilising research in the field of synthetic biology. Although in principle various different chemistries could be used for computation, the bottom-up approach to the creation of a minimal lifeform in synthetic biology will likely result in the creation of a simple regulatory network of genes, which could be used as an underlying platform for computation based on gene regulatory networks. Consideration was given to the possible amphiphiles and the characteristics of the vesicles resulting from the self-assembly process in Chapter 5. The amphiphiles considered in this chapter were representative of the two broad classes of possible amphiphiles which could be used for the creation of a computational vesicle. One of the most important aspects of the vesicle computer container will be the ability to filter the molecules which are able to enter and exit the membrane. A primitive example of this functionality was considered in Chapter 6, where the diffusion of matter from within a vesicle through simple pores in the membrane was investigated. The design of simple logic gates, built using gene regulatory style chemical interactions were investigated in detail in Chapter 7, and the results from this section indicate that simple logic gate behaviour can be encapsulated within a liposome. The computational experiments were extended in Chapter 8, in which vesicle computers were used to solve simple instances of the 2-SAT problem, in which the inputs to the calculation were encoded as chemical concentration gradients. The experiments showed that the transcriptional logic systems worked as intended and produced correct solutions to the problem.

To answer the second question, regarding how simulation and modelling techniques could aid the development of vesicle computing, this thesis has proposed and investigated the creation

of a multi-scale simulation framework, enabling the study of vesicle computation processes at two different scales, at the mesoscale with DPD, and the population or system scale with the stochastic simulation algorithm. Chapter 3 considers the different simulation techniques which are available and presents the framework along with the modular P systems language used for the specification of vesicle computations. Chapter 4 presented the development of a high performance implementation of DPD, including a novel algorithm for the implementation of DPD on Nvidia's CUDA framework, enabling the simulation of larger vesicle systems for a longer period of time. The simulation framework was then used for the experimentation performed in Chapters 5,6,7 and 8.

These questions have been addressed by the specification, development and application of a multi-scale simulation and modelling framework, utilising DPD, stochastic lattice population P systems specification and model checking. Simulations of several aspects which were perceived to be integral to the functioning of vesicle computing were performed at different scales and provide evidence for the key hypotheses of this work, that vesicles could be used to encapsulate reactions which result in logic gate based computation, and the construction of a multi-scale simulation and modelling framework can aid the design and reasoning process in synthetic biology. The contributions of each chapter towards this investigation are now summarised.

In Chapter 2, this work was placed within the context of the fields of unconventional computing and synthetic biology and the literature in these areas was reviewed. By surveying the latest developments in these fields, models of vesicle computing devices could be proposed which are of similar levels of complexity to those networks of gene expression which have been placed within vesicles in vitro. Existing research into the possibility of creating a protocell presents detailed information regarding the different amphiphiles, gene regulation networks and chemical reactions which have been successfully deployed within vesicles in the lab.

In Chapter 3 a novel multiscale simulation and modelling framework developed specifically for the investigation of vesicle computing systems was proposed. The approach combines detailed simulation of the membrane processes involved in vesicle computing using Dissipative Particle Dynamics, and investigation of longer term reaction dynamics over populations of vesicles using the stochastic simulation algorithm. Models are specified using a formal executable biology based syntax, stochastic P systems and lattice population P systems, which enable the development of vesicle computing models using a modular approach, consistent with the ambition in synthetic biology to create a library of genetic building blocks for synthetic systems, reducing the verbosity of the models and enabling reasoning about the design at a higher level of abstraction. The framework also permits the use of model checking of the vesicle computing models, meaning that model properties can be interrogated using continuous stochastic logic queries.

Chapter 4 presents the development of a key component of the vesicle computing simulation and modelling framework, the creation of a toolkit containing a highly optimised parallel implementation of the Dissipative particle dynamics, and auxiliary tool. DPD is a relatively computationally expensive simulation technique, due to the high level of detail with which models are expressed and

simulated. The high performance implementation enables rapid testing and prototyping of vesicle computing systems at a high level of detail. The chapter also described a collision based reaction scheme for DPD, so that the reaction dynamics of vesicle computing systems could be investigated, at least in a qualitative sense, using the technique. A correspondence between the rates of reaction in DPD and the Maxwell Boltzmann equation was discovered, and this enabled the conversion of reaction rates between DPD and SSA, simple models containing a single reaction were performed in both methods and good correspondence was found between the time series from both techniques.

Chapter 5 presented a literature review of the different amphiphile parameters used for bilayer and vesicle simulation, and amphiphile parameters were selected for the DPD simulations in the rest of this thesis. An analysis of a large number of DMPC vesicle formation experiments using the automated analysis methods provided by the DPD toolkit was also performed. The results of this analysis were the characterisation of the distribution of vesicle properties from over 160 vesicles and over 1000 micelles. The second aim of the work presented in Chapter 5 was the creation of a library of self-assembled membrane components, which could be composed and modified to create initial states for new simulations. In creating this library of vesicles and micelles, different membrane structures can be quickly assembled for the simulation of different vesicle computing systems.

In Chapter 6, a model of communication between vesicle computing elements was designed and simulated using both DPD and SSA. The experiments considered the rate of diffusion of the solvent encapsulated within a vesicle which contained pore inclusions within the membrane. The relationship between the rate of diffusion, and the number of pores contained within the membrane was investigated by simulation of the diffusion of particles across the membranes of vesicles with different numbers of pores contained within the membrane. A higher level model of the system was also created and simulated using SSA, by using the DPD results as a target for parameter estimation, permitted by the relatively computationally inexpensive nature of the SSA technique to reproduce the diffusion behaviour observed in the DPD simulation.

In Chapter 7, the dynamics of reaction systems reproducing the behaviour of basic logic gates was investigated using the multi-scale simulation framework. Several models of logic gates were created and encapsulated within the vesicles in DPD. The results illustrated in molecular scale simulation the catalytic effect of the vesicle membrane on reaction dynamics, and allowed the expression of simple logic functions within vesicles. The effect of different fluid immiscibilities on the rates of reaction was also considered. A model of a ring oscillator was adapted to function within DPD simulations timescales, and oscillations were shown to occur. A model checking based analysis of the logic gates enabled detailed characterisation of the connectivity properties, such as propagation delay.

In Chapter 8, behaviour of vesicle computations occurring across populations was considered using the lattice population P system functionality in the vesicle computing simulation framework. The computation of a solution to a pedagogical problem, the 2 variable Boolean satisfiability problem over populations of vesicles was shown, illustrating the encoding within the vesicles of more

complicated computations.

Some conclusions can be drawn from the execution of this study regarding both the proposed vesicle computing paradigm, and the simulation and modelling framework. The DPD and SSA techniques were complementary in the modelling of reactions in vesicle systems as it was concluded in Chapter 6 that the proposed DPD chemical reaction scheme allows analytical conversion of the rate parameters between the two techniques. However, the computationally expensive nature of DPD and the abstract nature of the membranes in stochastic P systems indicates that the selection of another simulation technique, targeting the length and time scales in between those which are appropriate for DPD and SSA may be a useful addition to the framework. The Smoldyn simulator described in Chapter 3 may be a good choice for this purpose as it combines desirable aspects of DPD, such as the ability to accurately model diffusion, and collision based reaction schemes, whilst dispensing with the simulation of the solvent particles which make DPD so computationally expensive, the technique also allows the specification of the shape of the membrane boundary, whereas in P systems and the stochastic simulation algorithm the membrane is an abstract concept simply representing the division of two compartments. The use of the lattice population and stochastic P systems was also shown to be a very powerful tool for specification of parameterised models, as evidenced in Chapter 8, where a successful model for solution to the 2-SAT problem was constructed in a programmatical fashion from composite elements. Furthermore, one of the key benefits of the DPD technique was the ability to demonstrate unexpected emergent properties in simulations of vesicle computing, such as the catalytic effect on chemical reactions that results from encapsulation within the the smaller vesicle volume and the effect of reactant solubilities on reaction rates, including the "bulging" effect on the structure of the vesicle (Chapter 7).

9.2 Future Work

Although this thesis has provided a computational study of several different aspects of vesicle computing, there are a number of areas which could be considered for the extension of the work on vesicle computing in the future.

9.2.1 A Roadmap for the Development of Vesicle Computation

The results presented in this thesis are an investigation into the possibility of using vesicles as the containers for the creation of cellular computers from the bottom up. The development of the vesicle computing concept and simulation and modelling techniques appropriate for its study have been the key focus of this work, as the creation of simulation and modelling techniques is useful from the point of view of generating new hypotheses regarding vesicle computing systems, and allows the designer of vesicle computers to encounter some of the problems which will arise in the creation of those systems before having to resort to costly and time consuming laboratory experiments. However, it is important to note that simulation and modelling techniques are tools which should be used in

the context of supporting laboratory experiment. This thesis therefore presents a roadmap for the implementation of vesicle computing, combined with a set of techniques and models which can be refined and modified as a result of laboratory experiments. In summary, one of the most important future goals of this research should be the further development of the models with regard to a chemical implementation. Such an implementation might proceed as follows:

1. The encapsulation of gene expression system within vesicles which produce simple computational functions. These simple functions could be in the form of the simple logic gates described in Chapter 7. This work will be performed in combination with development of new modular genetic components, such as biobricks, which will provide a coherent interface for the interconnection of gene expression systems.

2. The introduction of a crude metabolism into the computational automaton, capable of transforming externally available nutrients into the various different molecular components required for the system to maintain autopoiesis. Initially the vesicle computer might require that the majority of these nutrients (such as nucleotides, amino acids and ATP) are provided in the external environment, and enter the vesicle via diffusion through the membrane pores (i.e. like the Noireaux and Libchaber vesicle). However, the complexity of the metabolism could be slowly increased until the vesicle computing device is capable of manufacturing most of its required components internally.

3. The introduction of genes expressing a variety of different membrane proteins which embed within the membrane, and imbue the membrane with selective permeability, the ability to sense external chemical stimuli and help to maintain desirable properties of the membrane such as the tension and composition. The genes for these proteins could be taken from biological systems or designed from scratch.

4. The inclusion of simple chemical signalling systems, linked to the encapsulated gene expression systems. By including chemical signalling systems such as those found in quorum sensing bacteria, the vesicle computing devices can start to respond to environmental stimuli, and communicate with other vesicle computers.

5. Reproduction of vesicle computing devices with the aim of producing colonies of computational vesicles.

6. The differentiation of vesicle colony individuals based on chemical stimuli, i.e. the vesicles in the colony will alter their pathways of gene expression depending on the chemical signals present in the environment. This might allow vesicle computing elements to be specialised in certain functionalities. For example, some vesicle computers might dedicate themselves to producing nutrients which could be expelled into the surrounding medium, allowing other vesicles to focus energy and resources on the expression of proteins.

7. The formation of membrane substructures within the vesicle, which may be used for the storage or modification of nutrients, or for the encapsulation of functional modules which may contain proteins, RNA or other chemicals which might interfere with the primary vesicle computing systems.

8. The expression of motility proteins such as flagella, which will enable vesicle computing devices to relocate themselves in the environment, and travel along chemical gradients. For example a vesicle computer might travel towards the source of a glucose gradient, just as bacteria such as E. *coli* do.

Clearly, the roadmap for the creation of vesicle computing outlined above represents many years of laboratory work, and presents a large number of problems which as yet do not have known solutions. However, it is my hope that simulation and modelling techniques will be integral to the development of these vesicle computing devices at each stage, although this thesis presents simulation and modelling techniques which could aid the development of the first few stages of the implementation of vesicle computing, the aim is that these techniques will be improved upon and extended in a manner which is concurrent with the development of vesicle computers in the lab.

It becomes apparent that this incremental addition of functionality to the vesicle computing devices could be thought of as analogous to the evolutionary development of prokaryotic life. However, the difference between the way in which the development of vesicle computing devices should proceed, and the way in which evolution has resulted in the addition of new functionality in bacteria should be made clear: In creating these new vesicle computing designs, the behaviour of the system and the means by which it functions will always be well characterised and well understood, and so the task becomes one of engineering new functionality within an environment of manageable complexity, rather than one of reverse engineering an existing system which is not yet fully understood. Presumably once vesicle computing devices are able to reproduce, they will be subject to Darwinian evolution as it will be impossible to ensure that the copying of the vesicle computing components is entirely error free. The development of strategies to manage the evolutionary behaviour of vesicle computers so that they continue to function correctly will be a significant technological challenge.

9.2.2 The Further Development of the Vesicle Computing Framework

The future development of the vesicle computing framework can be approached from several directions. Firstly, there is the extension of the framework to include new simulation techniques. The work presented in this thesis has focused on two different length and timescales, the mesoscale with DPD, and the system (macro) scale with stochastic lattice population P systems. Although this enables simulations of most of the processes that are of interest to vesicle computing design, there are some aspects for which the DPD and SLPP systems are not well suited. Microscale processes are of interest in terms of providing accurate parameterisations of amphiphiles for DPD, and to study interactions between proteins and other molecules, a microscale simulation technique will therefore

become increasingly important in vesicle computing simulation. The addition of a traditional molecular dynamics software package to the framework would provide the necessary functionality. At the other end of the length and time scale continuum, the framework could be extended to consider the behaviour of populations of vesicle computers over evolutionary length and timescales. It may also be necessary to include new methods to study the coupling between the membrane dynamics and the encapsulated metabolism. An appropriate method may be something similar to multi-polar reactive DPD, or Schmoluchowski dynamics simulation.

Secondly, the development of the individual models used in this thesis to simulation vesicle properties can be considered. Although the parameters used in the simulations were derived from accurate models of the amphiphiles and reactions they represent, the true test of the accuracy of the simulation and modelling results would be to implement one of the vesicle computations modelled in previous chapters in the lab, and to make comparisons through experiment to determine whether the conclusions drawn from the simulation and modelling are correct. Initially, work in this area might include the application of the DPD and SLPP systems to the development of formation of vesicles from novel amphiphiles, and then proceed with the experimental verification of the diffusion properties of the vesicle membranes.

Many details of the underlying vesicle computing design, such as the metabolism and the transcriptional and translational systems were abstracted away from the DPD and SLPP models, in order to focus on the properties which were of interest. The design and inclusion of a simple proto-metabolism, which could be used to generate the necessary building blocks to maintain the vesicle computer and provide the nucleotides and amino acids required for gene expression would be a significant challenge both in terms of modelling and lab experiment. Alternative chemistries for the implementation of the logic gates could also be considered and may be more appropriate. The coupling of the reactions encapsulated within the vesicle core, and the vesicle membrane was not considered during this work, and so would be a good first aspect of the simulation to consider. Also, in extending the models it would be interesting to consider possible schemes for communication between vesicles as the spatial 2 SAT models in Chapter 8 only considered the response of vesicle computers to chemical stimuli from the environment. A model of this sort might show the interaction in the form of diffusive auto-inducer molecules between two vesicles in DPD or SSA.

When considering the "programming" of vesicle computing devices which are involved in chemical computation with one another, and more generally the development of computing devices according the amorphous and cell computing paradigms, it will be necessary to gain a better understanding of the design patterns and idioms which will be required to enable a designer to encode problems in the highly parallel heterogeneous vesicle computing environment. P systems and other concurrent formal languages such as π-calculus are a step in this direction, but mostly assume that the underlying hardware is static, and so it is likely that the development of vesicle computers will require techniques from other environments where the parties involved cannot be assumed to be always available and functioning correctly, such as peer to peer networking and network protocol

design.

Overall, the simulation and modelling framework which was proposed for the study of vesicle computation illustrated the benefits of a computational approach to the development and understanding of the hypothetical vesicle computing paradigm, in that it permitted the designer to encounter subtle difficulties and implementational details without resorting to laboratory experiments. However as with any simulation and modelling approach, the true test of its efficacy is in the ability to make valid predictions about physical systems. This work therefore presents a speculative first step along what will be a long and difficult avenue of research, culminating in the production of a fully artificial vesicle computer which will provide a platform for computing innovation in the approaching biotechnological era.

References

[1] http://www.ast.cam.ac.uk/ stg20/cuda/random/index.html.

[2] http://www.hyperrealm.com/libconfig/.

[3] H. Abelson, D. Allen, D. Coore, C. Hanson, G. Homsy, J. Thomas F. Knight, R. N. E. Rauch, G. J. Sussman, and R. Weiss. Amorphous computing. *Commun. ACM*, 43(5):74–82, 2000.

[4] A. Adamatzky and B. D. L. Costello. Experimental logical gates in a reaction-diffusion medium: The XOR gate and beyond. *Phys. Rev. E*, 66(4):046112, 2002.

[5] A. Adamatzky, B. D. L. Costello, and T. Asai. *Reaction-diffusion computers*. Elsevier Science Inc., 2005.

[6] A. Adamatzky, B. D. L. Costello, and L. Bull. On polymorphic logical gates in sub-excitable chemical medium. eprint arXiv:1007.0034, 2010.

[7] A. Adamatzky, B. De Lacey Costello, L. Bull, S. Stepney, and C. Teuscher. *Unconventional computing 2007*. Luniver Press, 2007.

[8] A. Adamatzky, J. Holley, L. Bull, and B. D. L. Costello. On computing in fine-grained compartmentalised belousov-zhabotinsky medium. eprint arXiv:1006.1900, 2010.

[9] A. Adamatzky and C. Teuscher. *From Utopian to Genuine Unconventional Computers*. Luniver Press, 2006.

[10] L. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266(5187):1021–1024, 1994.

[11] F. Ahmed and D. E. Discher. Self-porating polymersomes of PEG-PLA and PEG-PCL: hydrolysis-triggered controlled release vesicles. *J. Control. Release*, 96(1):37–53, 2004.

[12] A. Akthakul, C. E. Scott, A. M. Mayes, and A. J. Wagner. Lattice boltzmann simulation of asymmetric membrane formation by immersion precipitation. *J. Membrane Sci.*, 249(1-2):213–226, 2005.

[13] M. P. Allen and D. J. Tildesley. *Computer simulation of liquids.* Oxford University Press: New York, 1989.

[14] U. Alon. *An introduction to systems biology.* CRC Press, 2007.

[15] U. Alon. Network motifs: theory and experimental approaches. *Nat. Rev. Genet.*, 8:450 – 461, 2007.

[16] M. Amos, editor. *Cellular computing.* Oxford University Press, 2004.

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

[18] E. Andrianantoandro, S. Basu, D. K. Karig, and R. Weiss. Synthetic biology: New engineering rules for an emerging discipline. *Mol. Syst. Biol.*, 2, 2006.

[19] R. Arns. The other transistor: Early history of the metal-oxide semiconductor field-effect transistor. *Eng. Sci. Educ. J.*, 7:233 – 240, 1998.

[20] B. Aspvall, M. Plass, and R. Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Inform. Process. Lett.*, 8:121–123, 1979.

[21] G. Ayton, A. M. Smondyrev, S. G. Bardenhagen, P. McMurtry, and G. A. Voth. Interfacing molecular dynamics and macro-scale simulations for lipid bilayer vesicles. *Biophys. J.*, 83(2):1026–1038, 2002.

[22] G. Ayton and G. Voth. Bridging microscopic and mesoscopic simulations of lipid bilayers. *Biophys. J.*, 83(6):3357–3370, 2002.

[23] G. Ayton and G. Voth. Simulation of biomolecular systems at multiple length and time scales. *International Journal for Multiscale Computational Engineering*, 2, 2004.

[24] G. S. Ayton, J. L. McWhirter, P. McMurtry, and G. A. Voth. Coupling field theory with continuum mechanics: A simulation of domain formation in giant unilamellar vesicles. *Biophys. J.*, 88(6):3855–3869, 2005.

[25] G. S. Ayton and G. A. Voth. Multiscale simulation of transmembrane proteins. *J. Struct. Biol.*, 157(3):570–578, 2007.

[26] A. Aziz, K. Sanwal, V. Singhal, and R. Brayton. Model checking continuous time markov chains. *ACM T. Comput. Log.*, 1(1):162–170, 2000.

[27] C. Baier, B. Haverkort, H. Hermanns, and J. Katoen. Model-checking algorithms for continuous-time Markov chains. *IEEE T. Software Eng.*, 29(7):1–17, 2003.

[28] S. Basu, Y. Gerchman, C. H. Collins, F. H. Arnold, and R. Weiss. A synthetic multicellular system for programmed pattern formation. *Nature*, 434(7037):1130–1134, 2005.

[29] S. Basu, R. Mehreja, S. Thiberge, M.-T. Chen, and R. Weiss. Spatiotemporal control of gene expression with pulse-generating networks. *Proc. Natl. Acad. Sci. USA*, 101(17):6355–6360, 2004.

[30] G. Batt, C. Belta, and R. Weiss. Model checking genetic regulatory networks with parameter uncertainty. *Lect. Notes Comput. Sc.*, 4416:61–75, 2007.

[31] J. Baumgardner, K. Acker, O. Adefuye, S. Crowley, W. DeLoache, J. Dickson, L. Heard, A. Martens, N. Morton, M. Ritter, A. Shoecraft, J. Treece, M. Unzicker, A. Valencia, M. Waters, A. Campbell, L. Heyer, J. Poet, and T. Eckdahl. Solving a hamiltonian path problem with a bacterial computer. *J. Biol. Eng.*, 3(1):11, 2009.

[32] Y. Benenson, B. Gil, U. Ben-Dor, R. Adar, and E. Shapiro. An autonomous molecular computer for logical control of gene expression. *Nature*, 429(6990):423–429, 2004.

[33] Y. Benenson, T. Paz-Elizur, R. Adar, E. Keinan, Z. Livneh, and E. Shapiro. Programmable and autonomous computing machine made of biomolecules. *Nature*, 414(6862):430–434, 2001.

[34] S. A. Benner and A. M. Sismour. Synthetic biology. *Nat. Rev. Genet.*, 6(7):533–543, 2005.

[35] F. Bernardini, M. Gheorghe, and N. Krasnogor. Quorum sensing P systems. *Theor. Comput. Sci.*, 371(1-2):20–33, 2007.

[36] F. Bernardini, M. Gheorghe, N. Krasnogor, and G. Terrazas. Membrane computing current results and future problems. In S. B. Cooper, B. Lwe, and L. Torenvliet, editors, *New Computational Paradigms*, volume 3526 of *Lecture Notes in Computer Science*, pages 49–53. Springer Berlin / Heidelberg, 2005.

[37] G. Besold, I. Vattulainen, M. Karttunen, and J. M. Polson. Towards better integrators for dissipative particle dynamics simulations. *Phys. Rev. E*, 62(6):7611–7614, 2000.

[38] B. Bloom and C. Bancroft. Liposome-mediated biomolecular computation. In E. Winfree and D. K. Gifford, editors, *Proceedings of the 5th DIMACS Workshop on DNA Based Computers*, pages 39–48. American Mathematical Society, 1999.

[39] D. Boneh, C. Dunworth, R. J. Lipton, and J. Sgall. On the computational power of DNA. *Discrete Appl. Math.*, 71(1-3):79–94, 1996.

[40] C. Boyer and J. A. Zasadzinski. Multiple lipid compartments slow vesicle contents release in lipases and serum. *ACS Nano*, 1(3):176–182, 2007.

[41] R. S. Braich, N. Chelyapov, C. Johnson, P. W. K. Rothemund, and L. Adleman. Solution of a 20-variable 3-SAT problem on a DNA computer. *Science*, 296(5567):499–502, 2002.

[42] A. Buchanan, G. Gazzola, and M. A. Bed. Chapter 4 evolutionary design of a model of self-assembling chemical structures. In D. A. P. Natalio Krasnogor, Steven Gustafson and J. L. Verdegay, editors, *Systems Self-Assembly - Multidisciplinary Snapshots*, volume 5 of *Studies in Multidisciplinarity*, pages 79 – 100. Elsevier, 2008.

[43] J. Burch, E. Clarke, D. Long, K. Mcmillan, and D. Dill. Symbolic model checking for sequential circuit verification. *IEEE T. Comput. Aid. D.*, 13:401–424, 1993.

[44] J. Burch, E. Clarke, K. Mcmillan, D. Dill, and J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Inform. Comput.*, 98(2):142 – 170, 1992.

[45] C. Calude, J. Casti, and M. J. Dinneen, editors. *Unconventional Models of Computation*. Springer, 1998.

[46] B. Canton, A. Labno, and D. Endy. Refinement and standardization of synthetic biological parts and devices. *Nat. Biotechnol.*, 26(7):787–793, 2008.

[47] R. S. Cantor. Lipid composition and the lateral pressure profile in bilayers. *Biophys. J.*, 76(5):2625–2639, 1999.

[48] L. Cardelli. Brane calculi. *Computational Methods in Systems Biology*, pages 257–278, 2005.

[49] P. Cazzaniga, M. Gheorghe, N. K. an G. Mauri, D. Pescini, and F. Romero-Campero. *Probabilistic/stochastic models*, chapter 18. Oxford University Press, 2009.

[50] P. Cazzaniga, D. Pescini, F. J. Romero-Campero, D. Besozzi, and G. Mauri. Stochastic approaches in P systems for simulating biological systems. In *Proceedings of the Fourth Brainstorming Week on Membrane Computing*, 2006.

[51] G. Cevc. Lipid vesicles and other colloids as drug carriers on the skin. *Adv. Drug. Deliver. Rev.*, 56(5):675–711, 2004.

[52] I. A. Chen, K. Salehi-Ashtiani, and J. W. Szostak. RNA catalysis in model protocell vesicles. *J. Am. Chem. Soc.*, 127:13213–13219, 2005.

[53] S. Chen and G. D. Doolen. Lattice boltzmann method for fluid flow. *Annu. Rev. Fluid Mech.*, 30(1):329–364, 1998.

[54] S. Chen, C. Guo, G.-H. Hu, H.-Z. Liu, X.-F. Liang, J. Wang, J.-H. Ma, and L. Zheng. Dissipative particle dynamics simulation of gold nanoparticles stabilization by PEO-PPO-PEO block copolymer micelles. *Colloid Polym. Sci.*, 285:1543–1552, 2007.

[55] L. V. Chernomordik, J. Zimmerberg, and M. M. Kozlov. Membranes of the world unite! *J. Cell Biol.*, 175(2):201–207, 2006.

[56] H. Choi and C. Montemagno. Artificial organelle: ATP synthesis from cellular mimetic polymersomes. *Nano Lett.*, 5(12):2538–2542, 2005.

[57] H.-J. Choi, H. Lee, and C. D. Montemagno. Toward hybrid proteo-polymeric vesicles generating a photoinduced proton gradient for biofuel cells. *Nanotechnology*, 16(9):1589–1597, 2005.

[58] A. Church. An unsolvable problem of elementary number theory. *Am. J. Math.*, 58:345–363, 1936.

[59] F. Ciocchetta and J. Hillston. Process algebras in systems biology. *Lect. Notes Comput. Sc.*, 5016:265–312, 2008.

[60] E. Clarke, E. Emerson, and A. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM T. Progr. Lang. Sys.*, 8:244–263, 1986.

[61] J. Cohen. The crucial role of CS in systems and synthetic biology. *Commun. ACM*, 51(5):15–18, 2008.

[62] M. Cook, P. W. Rothemund, and E. Winfree. Self-assembled circuit patterns. *DNA Computing*, pages 1979–1979, 2004.

[63] L. Cronin, N. Krasnogor, B. G. Davis, C. Alexander, N. Robertson, J. H. G. Steinke, S. L. M. Schroeder, A. N. Khlobystov, G. Cooper, P. M. Gardner, P. Siepmann, B. J. Whitaker, and D. Marsh. The imitation game - a computational chemical approach to recognizing life. *Nat. Biotechnol.*, 24(10):1203–1206, 2006.

[64] A. Danchin. Bacteria as computers making computers. *FEMS Microbiol. Rev.*, 33(1):3–26, 2009.

[65] H. de Jong. Modeling and simulation of genetic regulatory systems: A literature review. *J. Comput. Biol.*, 9(1):67, 2002.

[66] D. Deamer. A giant step towards artificial life? *Trends Biotechnol.*, 23(7):336–338, 2005.

[67] D. Deamer, J. Dworkin, S. Sandford, M. Bernstein, and L. Allamandola. The first cell membranes. *Astrobiology*, 2(4), 2002.

[68] D. W. Deamer. Origins of life: How leaky were primitive cells? *Nature*, 454(7200):37–38, 2008.

[69] W. K. den Otter and J. H. R. Clarke. A new algorithm for dissipative particle dynamics. *Europhys. Lett.*, 53(4):426–431, 2001.

[70] D. Deutsch. Quantum theory, the Church-Turing principle and the universal quantum computer. *P. Roy. Soc. Lond. A Mat.*, 400(1818):97–117, 1985.

[71] A. DeVries, A. Mark, and S. Marrink. Molecular dynamics simulation of the spontaneous formation of a small DPPC vesicle in water in atomistic detail. *J. Am. Chem. Soc.*, 126(14):4488–4489, 2004.

[72] B. Di Ventura, C. Lemerle, K. Michalodimitrakis, and L. Serrano. From in vivo to in silico biology and back. *Nature*, 443(7111):527–533, 2006.

[73] S. P. Diggle, S. A. Crusz, and M. Cmara. Quorum sensing. *Curr. Biol.*, 17(21):907–910, 2007.

[74] B. M. Discher, Y.-Y. Won, D. S. Ege, J. C.-M. Lee, F. S. Bates, D. E. Discher, and D. A. Hammer. Polymersomes: Tough vesicles made from diblock copolymers. *Science*, 284(5417):1143–1146, 1999.

[75] D. E. Discher and A. Eisenberg. Polymer vesicles. *Science*, 297(5583):967–973, 2002.

[76] M. J. Doktycz and M. L. Simpson. Nano-enabled synthetic biology. *Mol. Syst. Biol.*, 3(125), 2007.

[77] J. Drouffe, A. Maggs, and S. Leibler. Computer simulations of self-assembled membranes. *Science*, 254(5036):1353–1356, 1991.

[78] D. A. Drubin, J. C. Way, and P. A. Silver. Designing biological systems. *Genes & Development*, 21(3):242–254, 2007.

[79] M. B. Elowitz and S. Leibler. A synthetic oscillatory network of transcriptional regulators. *Nature*, 403(6767):335–338, 2000.

[80] E. Emerson and E. Clarke. Using branching time temporal logic to synthesize synchronization skeletons. *Sci. Comput. Program.*, 2:241–266, 1982.

[81] D. Endy. Foundations for engineering biology. *Nature*, 438(7067):449–453, 2005.

[82] I. R. Epstein. Chemistry: Can droplets and bubbles think? *Science*, 315(5813):775–776, 2007.

[83] D. L. Ermak and J. A. McCammon. Brownian dynamics with hydrodynamic interactions. *J. Chem. Phys.*, 69(4):1352–1360, 1978.

[84] P. Español and P. Warren. Statistical-mechanics of dissipative particle dynamics. *Europhys. Lett.*, 30:191–196, 1995.

[85] R. Fassler, T. Hinze, T. Lenser, and P. Dittrich. Construction of oscillating chemical register machines on binary numbers using mass-action kinetics. In *Preliminary Proceedings Prague International Workshop on Membrane Computing in conjunction with DNA14*, 2008.

[86] H. Fellermann, S. Rasmussen, H.-J. Ziock, and R. V. Solé. Life cycle of a minimal protocell–a dissipative particle dynamics study. *Artif. Life*, 13(4):319–345, 2007.

[87] H. Fellermann and R. V. Solé. Minimal model of self-replicating nanocells: a physically embodied information-free scenario. *Philos. T. Roy. Soc. B*, 362(1486):1803–1811, 2007.

[88] J. Fisher and T. A. Henzinger. Executable cell biology. *Nat. Biotechnol.*, 25(11):1239–1249, 2007.

[89] P. J. Flory. Thermodynamics of high polymer solutions. *J. Chem. Phys.*, 10(1):51–61, 1942.

[90] L. R. Forrest and M. S. Sansom. Membrane simulations: bigger and better? *Curr. Opin. Struc. Biol.*, 10(2):174–181, 2000.

[91] A. C. Forster and G. M. Church. Towards synthesis of a minimal cell. *Mol. Syst. Biol.*, 2, 2006.

[92] R. Freund. Generalized P-systems. In *Fundamentals of Computation Theory: 12th International Symposium. Proceedings*, page 828, 1999.

[93] R. Freund, L. Kari, M. Oswald, and P. Sosik. Computationally universal P systems without priorities: two catalysts are sufficient. *Theor. Comput. Sci.*, 330(2):251–266, 2005.

[94] U. Frisch, B. Hasslacher, and Y. Pomeau. Lattice-gas automata for the navier-stokes equation. *Phys. Rev. Lett.*, 56(14):1505–1508, 1986.

[95] P. Frisco. The conformon-P system: a molecular and cell biology-inspired computability model. *Theor. Comput. Sci.*, 312(2-3):295–319, 2004.

[96] R. Füchslin, T. Maeke, and J. McCaskill. Spatially resolved simulations of membrane reactions and dynamics: Multipolar reaction DPD. *Eur. Phys. J. E*, 29(4):431–448, 2009.

[97] T. Gańti. *Chemoton Theory*. Kluwer Academic, 2004.

[98] L. Gao, J. Shillcock, and R. Lipowsky. Improved dissipative particle dynamics simulations of lipid bilayers. *J. Chem. Phys.*, 126(1):015101, 2007.

[99] P. M. Gardner, K. Winzer, and B. G. Davis. Sugar synthesis in a protocellular model leads to a cell signalling response in bacteria. *Nat. Chem.*, 1(5):377–383, 2009.

[100] T. S. Gardner, C. R. Cantor, and J. J. Collins. Construction of a genetic toggle switch in escherichia coli. *Nature*, 403(6767):339–342, 2000.

[101] A. Gehani and J. Reif. Micro flow bio-molecular computation. *Biosystems*, 52(1-3):197–216, 1999.

[102] D. G. Gibson, G. A. Benders, C. Andrews-Pfannkoch, E. A. Denisova, H. Baden-Tillson, J. Zaveri, T. B. Stockwell, A. Brownley, D. W. Thomas, M. A. Algire, C. Merryman, L. Young, V. N. Noskov, J. I. Glass, J. C. Venter, I. Hutchison, Clyde A., and H. O. Smith. Complete chemical synthesis, assembly, and cloning of a mycoplasma genitalium genome. *Science*, 319(5867):1215–1220, 2008.

[103] D. T. Gillespie. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *J. Comput. Phys.*, 22(4):403–434, 1976.

[104] D. T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *J. Phys. Chem.*, 81(25):2340–2361, 1977.

[105] D. T. Gillespie. Stochastic simulation of chemical kinetics. *Annu. Rev. Phys. Chem.*, 58(1):35–55, 2007.

[106] R. Goetz and R. Lipowsky. Computer simulations of bilayer membranes: Self-assembly and interfacial tension. *J. Chem. Phys.*, 108:7397–7409, 1998.

[107] D. Gonze, J. Halloy, and A. Goldbeter. Deterministic versus stochastic models for circadian rhythms. *J. Biol. Phys.*, 28(4):637–653, 2002.

[108] S. Green. CUDA particles. Technical report, NVIDIA Whitepaper, 2007.

[109] R. D. Groot and T. J. Madden. Dynamic simulation of diblock copolymer microphase separation. *J. Chem. Phys.*, 108(20):8713–8724, 1998.

[110] R. D. Groot, T. J. Madden, and D. J. Tildesley. On the role of hydrodynamic interactions in block copolymer microphase separation. *J. Chem. Phys.*, 110(19):9739–9749, 1999.

[111] R. D. Groot and K. L. Rabone. Mesoscopic simulation of cell membrane damage, morphology change and rupture by nonionic surfactants. *Biophys. J.*, 81:725–736, 2001.

[112] R. D. Groot and P. B. Warren. Dissipative particle dynamics: Bridging the gap between atomistic and mesoscopic simulation. *J. Chem. Phys.*, 107(11):4423–4435, 1997.

[113] R. Guantes and J. F. Poyatos. Dynamical principles of two-component genetic oscillators. *PLoS Comput. Biol.*, 2, 2006.

[114] C. C. Guet, M. B. Elowitz, W. Hsing, and S. Leibler. Combinatorial synthesis of genetic networks. *Science*, 296(5572):1466–1470, 2002.

[115] T. Gunnlaugsson, D. A. M. Donail, and D. Parker. Luminescent molecular logic gates: the two-input inhibit (inh) function. *Chem. Commun.*, (1):93–94, 2000.

[116] X. Guo, L. Zhang, Y. Qian, and J. Zhou. Effect of composition on the formation of poly(dl-lactide) microspheres for drug delivery systems: Mesoscale simulations. *Chem. Eng. J.*, 131(1-3):195–201, 2007.

[117] X. D. Guo, J. P. Tan, S. H. Kim, L. J. Zhang, Y. Zhang, J. L. Hedrick, Y. Y. Yang, and Y. Qian. Computational studies on self-assembled paclitaxel structures: Templates for hierarchical block copolymer assemblies and sustained drug release. *Biomaterials*, 30(33):6556–6563, 2009.

[118] C. K. Haluska, K. A. Riske, V. Marchi-Artzner, J.-M. Lehn, R. Lipowsky, and R. Dimova. Time scales of membrane fusion revealed by direct imaging of vesicle fusion with high temporal resolution. *Proc. Natl. Acad. Sci. USA*, 103(43):15841–15846, 2006.

[119] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal aspects of computing*, 6:512–535, 1994.

[120] J. Hardy and Y. Pomeau. Thermodynamics and hydrodynamics for a modeled fluid. *J. Math. Phys.*, 13(7):1042–1051, 1972.

[121] J. Hasty, D. McMillen, and J. J. Collins. Engineered gene circuits. *Nature*, 420(6912):224–230, 2002.

[122] K. A. Haynes, M. L. Broderick, A. D. Brown, T. L. Butner, J. O. Dickson, W. L. Harden, L. H. Heard, E. L. Jessen, K. J. Malloy, B. J. Ogden, S. Rosemond, S. Simpson, E. Zwack, A. M. Campbell, T. T. Eckdahl, L. J. Heyer, and J. L. Poet. Engineering bacteria to solve the burnt pancake problem. *J. Biol. Eng.*, 2, 2008.

[123] J. Heath, M. Kwiatkowska, G. Norman, D. Parker, and O. Tymchyshyn. Probabilistic model checking of complex biological pathways. *Theor. Comput. Sci.*, 391(3):239–257, 2008.

[124] M. Heinemann and S. Panke. Synthetic biology–putting engineering into biology. *Bioinformatics*, 22(22):2790–2799, 2006.

[125] M. Heiner, D. Gilbert, and R. Donaldson. Petri nets for systems and synthetic biology. *Lect. Notes Comput. Sc.*, 5016:215–264, 2008.

[126] A. Hjelmfelt, F. W. Schneider, and J. Ross. Pattern recognition in coupled chemical kinetic systems. *Science*, 260(5106):335–337, 1993.

[127] A. Hjelmfelt, E. D. Weinberger, and J. Ross. Chemical implementation of neural networks and Turing machines. *Proc. Natl. Acad. Sci. USA*, 88(24):10983–10987, 1991.

[128] A. Hjelmfelt, E. D. Weinberger, and J. Ross. Chemical implementation of finite-state machines. *Proc. Natl. Acad. Sci. USA*, 89(1):383–387, 1992.

[129] P. Hoogerbrugge and J. Koelman. Simulating microscopic hydrodynamic phenomena with dissipative particle dynamics. *Europhys. Lett.*, 19:155, 1992.

[130] J. E. Hopcroft. *Introduction to Automata Theory, Languages, and Computation*. Pearson Addison Wesley, 2007.

[131] M. L. Huggins. Some properties of solutions of long-chain compounds. *J. Phys. Chem.*, 46(1):151–158, 1942.

[132] M. L. Huggins. Theory of solutions of high polymers. *J. Am. Chem. Soc.*, 64(7):1712–1719, 1942.

[133] G. Illya, R. Lipowsky, and J. C. Shillcock. Effect of chain length and asymmetry on material properties of bilayer membranes. *J. Chem. Phys.*, 122(24):244901, 2005.

[134] G. Illya, R. Lipowsky, and J. C. Shillcock. Two-component membrane material properties and domain formation from dissipative particle dynamics. *J. Chem. Phys.*, 125(11):114710, 2006.

[135] A. Imparato, J. C. Shillcock, and R. Lipowsky. Shape fluctuations and elastic properties of two-component bilayer membranes. *Europhys. Lett.*, 69(4):650–656, 2005.

[136] A. F. Jakobsen, O. G. Mouritsen, and M. Weiss. Close-up view of the modifications of fluid membranes due to phospholipase A2. *J. Phys.: Condens. Matter*, 17(47):4015–4024, 2005.

[137] S. Ji. The cell as the smallest DNA-based molecular computer. *Biosystems*, 52(1-3):123–133, 1999.

[138] Y. Kim, M. Tewari, J. D. Pajerowski, S. Cai, S. Sen, J. Williams, S. Sirsi, G. Lutz, and D. E. Discher. Polymersome delivery of siRNA and antisense oligonucleotides. *J. Control. Release*, 134(2):132–140, 2009.

[139] T. Knight. Idempotent vector design for standard assembly of biobricks, 2003.

[140] H. Kobayashi, M. Krn, M. Araki, K. Chung, T. S. Gardner, C. R. Cantor, and J. J. Collins. Programmable cells: Interfacing natural and engineered gene networks. *Proc. Natl. Acad. Sci. USA*, 101(22):8414–8419, 2004.

[141] V. Kolisnychenko, G. Plunkett, C. D. Herring, T. Fehr, J. Psfai, F. R. Blattner, and G. Psfai. Engineering a reduced Escherichia coli genome. *Genome Res.*, 12(4):640–647, 2002.

[142] T. Konry and D. R. Walt. Intelligent medical diagnostics via molecular logic. *J. Am. Chem. Soc.*, 131(37):13232–13233, 2009.

[143] M. Kranenburg, C. Laforge, and B. Smit. Mesoscopic simulations of phase transitions in lipid bilayers. *Physical Chemistry Chemical Physics*, 6:4531–4534, 2004.

[144] M. Kranenburg, J.-P. Nicolas, and B. Smit. Comparison of mesoscopic phospholipid-water models. *Physical Chemistry Chemical Physics*, 6(16):4142–4151, 2004.

[145] M. Kranenburg and B. Smit. Simulating the effect of alcohol on the structure of a membrane. *FEBS Lett.*, 568(1-3):15–18, 2004.

[146] M. Kranenburg, M. Venturoli, and B. Smit. Phase behavior and induced interdigitation in bilayers studied with dissipative particle dynamics. *J. Phys. Chem. B*, 107(41):11491–11501, 2003.

[147] G. Kresse and J. Hafner. Ab initio molecular dynamics for liquid metals. *Phys. Rev. B*, 47(1):558–561, 1993.

[148] M. Kwiatkowska, G. Norman, and D. Parker. PRISM: Probabilistic symbolic model checker. *Computer Performance Evaluation: Modelling Techniques and Tools*, pages 113–140, 2002.

[149] M. Kwiatkowska, G. Norman, and D. Parker. Using probabilistic model checking in systems biology. *ACM SIGMETRICS Performance Evaluation Review*, 35(4):14–21, 2008.

[150] M. Laradji and P. B. S. Kumar. Dynamics of domain growth in self-assembled fluid vesicles. *Phys. Rev. Lett.*, 93:198105, 2004.

[151] M. Laradji and P. B. S. Kumar. Domain growth, budding, and fission in phase-separating self-assembled fluid bilayers. *J. Chem. Phys.*, 123(22):224902, 2005.

[152] M. Laradji and P. B. S. Kumar. Anomalously slow domain growth in fluid membranes with asymmetric transbilayer lipid distribution. *Phys. Rev. E*, 73(4):040901, 2006.

[153] W. Lee, S. Ju, Y. Wang, and J. Chang. Modeling of polyethylene and poly (L-lactide) polymer blends and diblock copolymer: Chain length and volume fraction effects on structural arrangement. *J. Chem. Phys.*, 127(6):064902, 2007.

[154] D. Li and X. Y. Liu. Examination of membrane fusion by dissipative particle dynamics simulation and comparison with continuum elastic models. *J. Chem. Phys.*, 122(17):174909, 2005.

[155] H. B. Li, H. H. Yi, X. W. Shan, and H. P. Fang. Shape changes and motion of a vesicle in a fluid using a lattice boltzmann model. *Europhys. Lett.*, 81(5):54002, 2008.

[156] L. Li, P. Siepmann, J. Smaldon, G. Terrazas, and N. Krasnogor. Chapter 13 automated self-assembling programming. In D. A. P. Natalio Krasnogor, Steven Gustafson and J. L. Verdegay, editors, *Systems Self-Assembly - Multidisciplinary Snapshots*, volume Volume 5, pages 281–307. Elsevier, 2008.

[157] X. Li, J. Guo, Y. Liu, and H. Liang. Microphase separation of diblock copolymer poly(styrene-b-isoprene): A dissipative particle dynamics simulation study. *J. Chem. Phys.*, 130(7):074908, 2009.

[158] E. Liberman. Analog-digital molecular cell computer. *Biosystems*, 11(2-3):111–124, 1979.

[159] E. Lindahl, B. Hess, and D. van der Spoel. GROMACS 3.0: a package for molecular simulation and trajectory analysis. *J. Mol. Model.*, 7(8):306–317, 2001.

[160] K. Lipkow, S. Andrews, and D. Bray. Simulated diffusion of phosphorylated chey through the cytoplasm of escherichia coli. *J. Bacteriol.*, 187:23–25, 2005.

[161] K. Lipkow and D. Odde. Model for protein concentration gradients in the cytoplasm. *Cell. Mol. Bioeng.*, 1(1):84–92, 2008.

[162] R. Lipton. DNA solution of hard computational problems. *Science*, 268(5210):542–545, 1995.

[163] H. Lodish, A. Berk, C. Kaiser, M. Krieger, M. Scott, A. Bretscher, H. Ploegh, and P. Matsudaira. *Molecular Cell Biology.* W.H.Freeman & Co Ltd, 2007.

[164] H. Lomas, I. Canton, S. MacNeil, J. Du, S. P. Armes, A. J. Ryan, A. L. Lewis, and G. Battaglia. Biomimetic pH sensitive polymersomes for efficient DNA encapsulation and delivery. *Advanced Materials*, 19(23):4238–4243, 2007.

[165] D. Loss and D. P. DiVincenzo. Quantum computation with quantum dots. *Phys. Rev. A*, 57(1):120–126, 1998.

[166] C. P. Lowe. An alternative approach to dissipative particle dynamics. *Europhys. Lett.*, 47(2):145–151, 1999.

[167] Z. Luo and J. Jiang. Molecular dynamics and dissipative particle dynamics simulations for the miscibility of poly(ethylene oxide)/poly(vinyl chloride) blends. *Polymer*, 51(1):291–299, 2010.

[168] J. Macia and R. V. Solé. Protocell self-reproduction in a spatially extended metabolism-vesicle system. *J. Theor. Biol.*, 245(3):400–410, 2007.

[169] J. Macia and R. V. Solé. Synthetic turing protocells: vesicle self-reproduction through symmetry-breaking instabilities. *Philos. T. Roy. Soc. B*, 362(1486):1821–1829, 2007.

[170] D. Madina, N. Ono, and T. Ikegami. Cellular evolution in a 3D lattice artificial chemistry. *Advances in Artificial Life*, pages 59–68, 2003.

[171] M. O. Magnasco. Chemical kinetics is turing universal. *Phys. Rev. Lett.*, 78(6):1190, 1997.

[172] D. C. Magri. A fluorescent and logic gate driven by electrons and protons. *New J. Chem.*, 33(3):457–461, 2009.

[173] S. S. Mansy, J. P. Schrum, M. Krishnamurthy, S. Tobe, D. A. Treco, and J. W. Szostak. Template-directed synthesis of a genetic polymer in a model protocell. *Nature*, 454(7200):122–125, 2008.

[174] V. S. Markin and J. P. Albanesi. Membrane fusion: Stalk model revisited. *Biophys. J.*, 82(2):693–712, 2002.

[175] S. Marrink and A. Mark. The mechanism of vesicle fusion as revealed by molecular dynamics simulations. *J. Am. Chem. Soc.*, 125(37):11144–11145, 2003.

[176] S. Marrink and A. Mark. Molecular dynamics simulation of the formation, structure, and dynamics of small phospholipid vesicles. *J. Am. Chem. Soc.*, 125(49):15233–15242, 2003.

[177] S. J. Marrink, A. H. de Vries, and A. E. Mark. Coarse grained model for semiquantitative lipid simulations. *J. Phys. Chem. B*, 108(2):750–760, 2004.

[178] S. J. Marrink and A. E. Mark. The mechanism of vesicle fusion as revealed by molecular dynamics simulations. *J. Am. Chem. Soc.*, 125(37):11144–11145, 2003.

[179] G. Marsaglia. Random number generators. *J. Mod. App. Stat. Meth.*, 2(1):2–13, 2003.

[180] G. Marsaglia. Seeds for random number generators. *Commun. ACM*, 46(5):90–93, 2003.

[181] F. Mavelli and K. Ruiz-Mirazo. Stochastic simulations of minimal self-reproducing cellular systems. *Philos. T. Roy. Soc. B*, 362(1486):1789–1802, 2007.

[182] J. M. McCollum, G. D. Peterson, C. D. Cox, M. L. Simpson, and N. F. Samatova. The sorting direct method for stochastic simulation of biochemical systems with varying reaction execution behavior. *Comput. Biol. Chem.*, 30(1):39–49, 2006.

[183] F. Meng, Z. Zhong, and J. Feijen. Stimuli-responsive polymersomes for programmed drug delivery. *Biomacromolecules*, 10(2):197–209, 2009.

[184] K. Michielsen and H. De Raedt. Integral-geometry morphological image analysis. *Phys. Rep.*, 347:461–538, 2001.

[185] H. Minkowski. Volumen und oberflche. *Math. Ann.*, 57(4):447–495, 1903.

[186] P.-A. Monnard, A. Luptak, and D. W. Deamer. Models of primitive cellular life: polymerases and templates in liposomes. *Philos. T. Roy. Soc. B*, 362(1486):1741–1750, 2007.

[187] M. Motornov, J. Zhou, M. Pita, V. Gopishetty, I. Tokarev, E. Katz, and S. Minko. "chemical transformers" from nanoparticle ensembles operated with logic. *Nano Lett.*, 8(9):2993–2997, 2008.

[188] S. Muramatsu, K. Kinbara, H. Taguchi, N. Ishii, and T. Aida. Semibiological molecular machine with an implemented "and" logic gate for regulation of protein folding. *J. Am. Chem. Soc.*, 128(11):3764–3769, 2006.

[189] M. Nakajima, K. Imai, H. Ito, T. Nishiwaki, Y. Murayama, H. Iwasaki, T. Oyama, and T. Kondo. Reconstitution of circadian oscillation of cyanobacterial KaiC phosphorylation in vitro. *Science*, 308(5720):414–415, 2005.

[190] M. Nallani, S. Benito, O. Onaca, A. Graff, M. Lindemann, M. Winterhalter, W. Meier, and U. Schwaneberg. A nanocompartment system (synthosome) designed for biotechnological applications. *J. Biotechnol.*, 123(1):50–59, 2006.

[191] J. V. Neumann. *Theory of Self-Reproducing Automata*. University of Illinois Press, Champaign, IL, USA, 1966.

[192] P. Nikunen, M. Karttunen, and I. Vattulainen. How would you integrate the equations of motion in dissipative particle dynamics simulations? *Comput. Phys. Commun.*, 153:407, 2003.

[193] M. Nixon and A. S. Aguado. *Feature extraction and image processing*. Academic Press, 2007.

[194] H. Noguchi. Polyhedral vesicles: A brownian dynamics simulation. *Phys. Rev. E*, 67(4):041901, 2003.

[195] H. Noguchi and M. Takasu. Self-assembly of amphiphiles into vesicles: A brownian dynamics simulation. *Phys. Rev. E*, 64(4):41913, 2001.

[196] H. Noguchi and M. Takasu. Adhesion of nanoparticles to vesicles: A brownian dynamics simulation. *Biophys. J.*, 83(1):299–308, 2002.

[197] H. Noguchi and M. Takasu. Structural changes of pulled vesicles: A brownian dynamics simulation. *Phys. Rev. E*, 65(5):051907, 2002.

[198] V. Noireaux and A. Libchaber. A vesicle bioreactor as a step toward an artificial cell assembly. *Proc. Natl. Acad. Sci. USA*, 101(51):17669–17674, 2004.

[199] S. Nomura, K. Tsumoto, T. Hamada, K. Akiyoshi, Y. Nakatani, and K. Yoshikawa. Gene expression within cell-sized lipid vesicles. *ChemBioChem*, 4(11):1172–1175, 2003.

[200] P. Nurse. Life, logic and information. *Nature*, 454(7203):424–426, 2008.

[201] T. Oberholzer, K. H. Nierhaus, and P. L. Luisi. Protein expression in liposomes. *Biochem. Bioph. Res. Co.*, 261(2):238–241, 1999.

[202] M. Oltean and O. Muntean. Solving NP-Complete problems with delayed signals: An overview of current research directions. In *Optical SuperComputing: First International Workshop, OSC 2008, Vienna, Austria, August 26, 2008, Proceedings*, pages 115–127, 2008.

[203] N. Ono and T. Ikegami. Artificial chemistry: Computational studies on the emergence of self-reproducing units. *Lect. Notes Comput. Sc.*, 2159:186, 2001.

[204] V. Ortiz, S. Nielsen, D. Discher, M. Klein, R. Lipowsky, and J. Shillcock. Dissipative particle dynamics simulations of polymersomes. *J. Phys. Chem. B*, 109(37):17708–17714, 2005.

[205] J. Owens, M. Houston, D. Luebke, S. Green, J. Stone, and J. Phillips. GPU computing. *Proceedings of the IEEE*, 96(5):879–899, 2008.

[206] I. Pagonabarraga, M. H. J. Hagen, and D. Frenkel. Self-consistent dissipative particle dynamics algorithm. *Europhys. Lett.*, 42:377–382, 1998.

[207] C. H. Papadimitriou. *Computational Complexity*. Addison Wesley, 1994.

[208] D. Parker. *Implementation of Symbolic Model Checking for Probabilistic Systems*. PhD thesis, University of Birmingham, 2002.

[209] G. Pasparakis and C. Alexander. Sweet talking double hydrophilic block copolymer vesicles. *Angew. Chem. Int. Ed.*, 49(2):241, 2008.

[210] G. Paun. Computing with membranes. Technical report, 1998.

[211] G. Paun. *Membrane Computing: An Introduction*. Springer-Verlag New York, Inc., 2002.

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

[213] S. Plimpton and B. Hendrickson. A new parallel method for molecular dynamics simulation of macromolecular systems. *J. Comput. Chem.*, 17(3):326–337, 1996.

[214] A. Pohorille and D. Deamer. Artificial cells: prospects for biotechnology. *Trends Biotechnol.*, 20(3):123–128, 2002.

[215] G. Posfai, I. Plunkett, Guy, T. Feher, D. Frisch, G. M. Keil, K. Umenhoffer, V. Kolisnychenko, B. Stahl, S. S. Sharma, M. de Arruda, V. Burland, S. W. Harcum, and F. R. Blattner. Emergent properties of reduced-genome escherichia coli. *Science*, 312(5776):1044–1046, 2006.

[216] M. Prakash and N. Gershenfeld. Microfluidic Bubble Logic. *Science*, 315(5813):832–835, 2007.

[217] C. Priami. Algorithmic systems biology. *Commun. ACM*, 52(5):80–88, 2009.

[218] S. M. Rafelski and W. F. Marshall. Building the cell: design principles of cellular architecture. *Nat. Rev. Mol. Cell. Biol.*, 9(8):593–602, 2008.

[219] D. C. Rapaport. *The Art of Molecular Dynamics Simulation*. Cambridge University Press, 1998.

[220] S. Rasmussen, M. A. Bedau, L. Chen, D. Deamer, D. C. Krakauer, N. H. Packard, and P. F. Stadler, editors. *Protocells: Bridging Nonliving and Living Matter*. The MIT Press, 2008.

[221] F. J. Romero-Campero, H. Cao, M. Camara, and N. Krasnogor. Structure and parameter estimation for cell systems biology models. In *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 331–338, New York, NY, USA, 2008. ACM.

[222] F. J. Romero-Campero and M. J. Perez-Jimenez. Modelling gene expression control using P systems: The Lac Operon, a case study. *Biosystems*, 91(3):438–457, 2008.

[223] F. J. Romero-Campero, J. Twycross, M. Camara, M. Bennett, M. Gheorghe, and N. Krasnogor. Modular assembly of cell systems biology models using p systems. *Internat. J. Found. Comput. Sci.*, 20(3):427–442, 2009.

[224] R. Roodbeen and J. C. M. van Hest. Synthetic cells and organelles: compartmentalization strategies. *BioEssays*, 31(12):1299–1308, 2009.

[225] R. E. Rudd and J. Q. Broughton. Coarse-grained molecular dynamics and the atomic limit of finite elements. *Phys. Rev. B*, 58(10):5893–5896, 1998.

[226] K. Ruiz-Mirazo and F. Mavelli. On the way towards 'basic autonomous agents': Stochastic simulations of minimal lipid-peptide cells. *Biosystems*, 91(2):374–387, 2008.

[227] A. Sawchuk and T. Strand. Digital optical computing. *Proc. IEEE*, 72:758– 779, 1984.

[228] L. Serrano. Synthetic biology: Promises and challenges. *Mol. Syst. Biol.*, 3, 2007.

[229] E. Shapiro and Y. Benenson. Bringing DNA computers to life. *Scientific American*, 294:44–51, 2006.

[230] R. Shetty, D. Endy, and T. Knight. Engineering BioBrick vectors from BioBrick parts. *J. Biol. Eng.*, 2(1):5, 2008.

[231] J. Shillcock and R. Lipowsky. Dissipative particle dynamics simulations of planar amphiphilic bilayers. In *NIC Symposium 2001, Proceedings*, 2001.

[232] J. Shillcock and R. Lipowsky. Tension-induced fusion of bilayer membranes and vesicles. *Nature Mater.*, 4:225–228, 2005.

[233] J. C. Shillcock and R. Lipowsky. Equilibrium structure and lateral stress distribution of amphiphilic bilayers from dissipative particle dynamics simulations. *J. Chem. Phys.*, 117:5048–5061, 2002.

[234] J. C. Shillcock and R. Lipowsky. The computational route from bilayer membranes to vesicle fusion. *J. Phys.: Condens. Matter*, 18(28):1191–1219, 2006.

[235] J. C. Shillcock and U. Seifert. Thermally induced proliferation of pores in a model fluid membrane. *Biophys. J.*, 74:1754–1766, 1998.

[236] N. M. Shnerb, Y. Louzoun, E. Bettelheim, and S. Solomon. The importance of being discrete: Life always wins on the surface. *Proc. Natl. Acad. Sci. USA*, 97(19):10322–10324, 2000.

[237] P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Rev.*, 41(2):303–332, 1999.

[238] H. T. Siegelmann. *Neural networks and analog computation: beyond the Turing limit*. Birkhauser Boston Inc., Cambridge, MA, USA, 1999.

[239] R. Silva-Rocha and V. de Lorenzo. Mining logic gates in prokaryotic transcriptional regulation networks. *FEBS Lett.*, 582(8):1237–1244, 2008.

[240] M. L. Simpson, G. S. Sayler, J. T. Fleming, and B. Applegate. Whole-cell biocomputing. *Trends Biotechnol.*, 19(8):317–323, 2001.

[241] J. S. Sims and N. S. Martys. Simulation of sheared suspensions with a parallel implementation of QDPD. *J. Res. Natl. Inst. Stan.*, 109:277, 2004.

[242] M. Sipper. The emergence of cellular computing. *Computer*, 32(7):18–26, 1999.

[243] J. Smaldon, F. Romero-Campero, J. Blakes, J. Twycross, and N. Krasnogor. A cellular computing based solver for instances of the 2-SAT problem. *J. Logic Comput.*, Submitted, 2009.

[244] R. G. Smith, N. D'Souza, and S. Nicklin. A review of biosensors and biologically-inspired systems for explosives detection. *The Analyst*, 133(5):571–584, 2008.

[245] D. Soloveichik, G. Seelig, and E. Winfree. DNA as a universal substrate for chemical kinetics. *Proc. Natl. Acad. Sci. USA*, 107(42):57–69, 2009.

[246] R. V. Sol, A. Munteanu, C. Rodriguez-Caso, and J. Maca. Synthetic protocell biology: from reproduction to computation. *Philos. T. Roy. Soc. B*, 362(1486):1727–1739, 2007.

[247] L. Song, M. R. Hobaugh, C. Shustak, S. Cheley, H. Bayley, and J. E. Gouaux. Structure of staphylococcal α-Hemolysin, a heptameric transmembrane pore. *Science*, 274(5294):1859–1865, 1996.

[248] P. Steve and H. Bruce. *Parallel Molecular Dynamics Algorithms for Simulation of Molecular Systems*, chapter 10, pages 114–132.

[249] M. N. Stojanovic and D. Stefanovic. A deoxyribozyme-based molecular automaton. *Nat. Biotech.*, 21(9):1069–1074, 2003.

[250] J. Stricker, S. Cookson, M. R. Bennett, W. H. Mather, L. S. Tsimring, and J. Hasty. A fast, robust and tunable synthetic gene oscillator. *Nature*, 456(7221):516–519, 2008.

[251] E. Szathmary. Life: In search of the simplest cell. *Nature*, 433(7025):469–470, 2005.

[252] J. W. Szostak, D. P. Bartel, and P. L. Luisi. Synthesizing life. *Nature*, 409(6818):387–390, 2001.

[253] K. Takakura, T. Toyota, and T. Sugawara. A novel system of self-reproducing giant vesicles. *J. Am. Chem. Soc.*, 125(27):8134–8140, 2003.

[254] C. Talcott and D. Dill. Multiple representations of biological processes. *Transactions on Computational Systems Biology*, 6:221–245, 2006.

[255] G. Terrazas and N. Krasnogor. Automated tile design for self-assembly conformations. In *Proceedings of the IEEE Congress on Evolutionary Computation*, 2005.

[256] S. F. M. van Dongen, M. Nallani, J. J. L. M. Cornelissen, R. J. M. Nolte, and J. C. M. van Hest. A three-enzyme cascade reaction through positional assembly of enzymes in a polymersome nanoreactor. *Chemistry - A European Journal*, 15(5):1107–1114, 2009.

[257] I. Vattulainen, M. Karttunen, G. Besold, and J. Polson. Integration schemes for dissipative particle dynamics simulations: From softly interacting systems towards hybrid models. *J. Chem. Phys.*, 116:3967, 2002.

[258] B. Venturoli, M.and Smit. Simulating the self-assembly of model membranes. *Phys. Chem. Comm.*, 2:45–49, 1999.

[259] C. A. Voigt. Genetic parts to program bacteria. *Current Opinion in Biotechnology*, 17(5):548–557, 2006.

[260] J. von Neumann. First draft of a report on the edvac. *IEEE Ann. Hist. Comput.*, 15(4):27–75, 1993.

[261] R. Wang and L. Chen. Synchronizing genetic oscillators by signaling molecules. *J. Biol. Rhythm.*, 20(3):257–269, 2005.

[262] R. Weiss and S. Basu. The device physics of cellular logic gates. In *The First Workshop on Non Silicon Computing*, 2002.

[263] R. Weiss, S. Basu, S. Hooshangi, A. Kalmbach, D. Karig, R. Mehreja, and I. Netravali. Genetic circuit building blocks for cellular computation, communications, and signal processing. *Natural Computing*, 2(1):47–84, 2003.

[264] R. Weiss, G. E. Homsy, T. F. Knight, and Jr. Toward in vivo digital circuits. In *Proceedings of the Dimacs Workshop on Evolution as Computation*, 1999.

[265] R. Weiss and T. Knight. Engineered communications for microbial robotics. *Lect. Notes Comput. Sc.*, 2054:1–16, 2001.

[266] D. J. Wilkinson. *Stochastic modelling for systems biology.* CRC Press, 2006.

[267] M. R. Wilson, M. P. Allen, M. A. Warren, A. Sauron, and W. Smith. Replicated data and domain decomposition molecular dynamics techniques for simulation of anisotropic potentials. *J. Comput. Chem.*, 18(4):478–488, 1997.

[268] E. Winfree. *Algorithmic self-assembly of DNA*. PhD thesis, California Institute of Technology, 1998.

[269] E. Winfree, F. Liu, L. A. Wenzler, and N. C. Seeman. Design and self-assembly of two-dimensional DNA crystals. *Nature*, 394(6693):539–544, 1998.

[270] S. Yamamoto and S. Hyodo. Budding and fission dynamics of two-component vesicles. *J. Chem. Phys.*, 118:7937–7943, 2003.

[271] S. Yamamoto, Y. Maruyama, and S. aki Hyodo. Erratum: "dissipative particle dynamics study of spontaneous vesicle formation of amphiphilic molecules". *J. Chem. Phys.*, 117(6):2990–2990, 2002.

[272] S. Yamamoto, Y. Maruyama, and S.-A. Hyodo. Dissipative particle dynamics study of spontaneous vesicle formation of amphiphilic molecules. *J. Chem. Phys.*, 116:5842–5849, 2002.

[273] C. Zandron, C. Ferretti, and G. Mauri. Solving NP-Complete problems using P systems with active membranes. In *UMC 2000: Proceedings of the Second International Conference on Unconventional Models of Computation*, pages 289–301, London, UK, 2000. Springer-Verlag.

[274] Y. Zhao, L.-Y. You, Z.-Y. Lu, and C.-C. Sun. Dissipative particle dynamics study on the multicompartment micelles self-assembled from the mixture of diblock copolymer poly(ethyl ethylene)-block-poly(ethylene oxide) and homopolymer poly(propylene oxide) in aqueous solution. *Polymer*, 50(22):5333–5340, 2009.

Appendix A

Example Configuration File for the DPD Simulator

The following configuration file configures a volume which is $30^3 r_c$ and contains 20% DMPC polymers by volume fraction.

```
system:
{
    rho = 3.0;
    sigma = 3.0;
    gamma = 4.5;

    randomSeed = -1;

    sidelengths: { x = 30.0; y = 30.0; z = 30.0; };

    equilibriate = false;
    equilibriateSteps = 1000;
    equilibriatedt = 0.05;

    changeType = false;
    changeFromType = [1];
    changeToType = [2];
    changeTypeStep = [1000];
};

data:
{
    saveinterval = 100;
    savestartstep = 0;
    saveendstep = -1;
    statfile = "out.stats";
    statinterval = 20;
    statstartstep = -1;
};

integration:
{
    dt = 0.05;
    lambda = 0.65;
    numsteps = 100000;
};

particletypes:
{
    typenames = ["water", "oil", "head"];
    percentages = [0.0, 0.0, 0.0];

    alphamatrix = (
                        [78.00, 104.00, 75.80],
                        [104.00, 78.00, 104.00],
                        [75.80, 104.00, 86.7]
                    );
};

polymertypes:
```

```
51  {
52      typenames = ["DMPC"];
53      percentages = [20.0];
54      monomerTypes = ([2,2,2,1,1,1,1,1,1,1,1,1]);
55      bondMaps = (([0,1],[1,2],[2,3],[3,4],[4,5],[5,6],[6,7],[2,8],[8,9],[9,10],[10,11],[11,12]));
56      bondStrengths = ([100.00, 100.00, 100.00, 100.00,100.00,100.00,100.00,100.00,100.00,100.00,100.00,100.00]);
57      bondLengths = ([0.7, 0.7, 0.7, 0.7, 0.7, 0.7, 0.7, 0.7, 0.7, 0.7, 0.7, 0.7]);
58      angleMaps = (([3,2,8], [3,4,5], [4,5,6], [5,6,7], [8,9,10], [9,10,11], [10,11,12]));
59      angleStrengths = ([6.0, 6.0, 6.0, 6.0, 6.0, 6.0, 6.0]);
60      angleAngles = ([1.57079632679, 3.14159265359, 3.14159265359, 3.14159265359, 3.14159265359, 3.14159265359, 3.14159265359]);
61      monomerPositions = (([0.0, 4.0, 0.0], [0.0, 3.0, 0.0], [0.0, 2.0, 0.0],
62                          [-0.5, 1.0, 0.0], [-0.5, 0.0, 0.0], [-0.5, -1.0, 0.0],
63                          [-0.5, -2.0, 0.0], [-0.5, -3.0, 0.0], [0.5, 1.0, 0.0],
64                          [0.5, 0.0, 0.0], [0.5, -1.0, 0.0], [0.5, -2.0, 0.0], [0.5, -3.0, 0.0]));
65      #increase/decrease to grow/shrink the polymer
66      monomerPositionScalar = [0.7];
67  };
```

Appendix B

Stereoscopic Images of Simulated Vesicles

This appendix contains cross-eyed stereoscopic images of some of the vesicles which were used in this thesis, the images were generated by exporting polymer position data from simulation data files to Povray format, using the *dpdanalysis* tool. The images were then rendered using the Povray raytracing application.

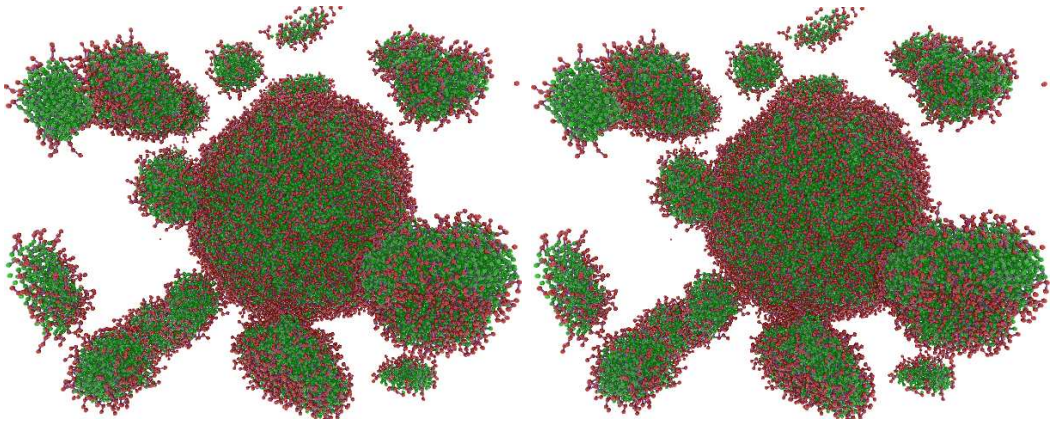


Figure B.1: Vesicles and Micelles formed from DMPC amphiphiles

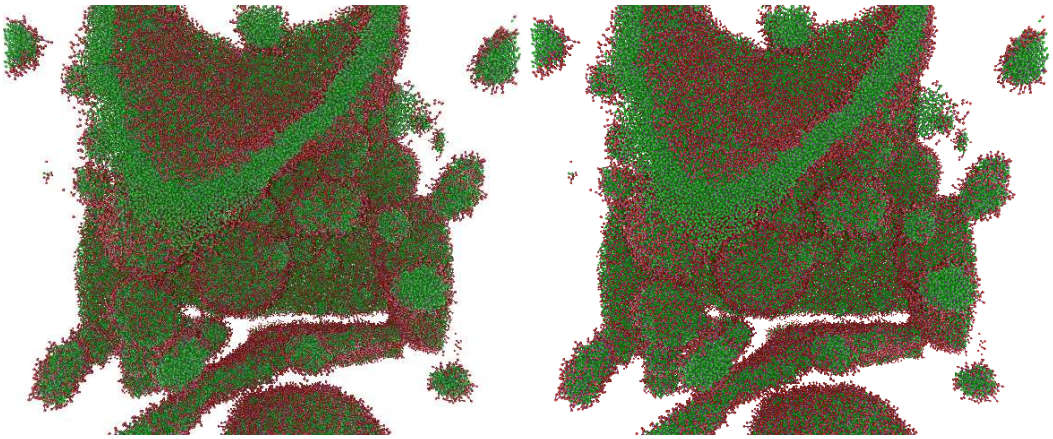


Figure B.2: A larger simulation of self-assembly from DMPC, containing vesicles, micelles and flat patches of bilayer.

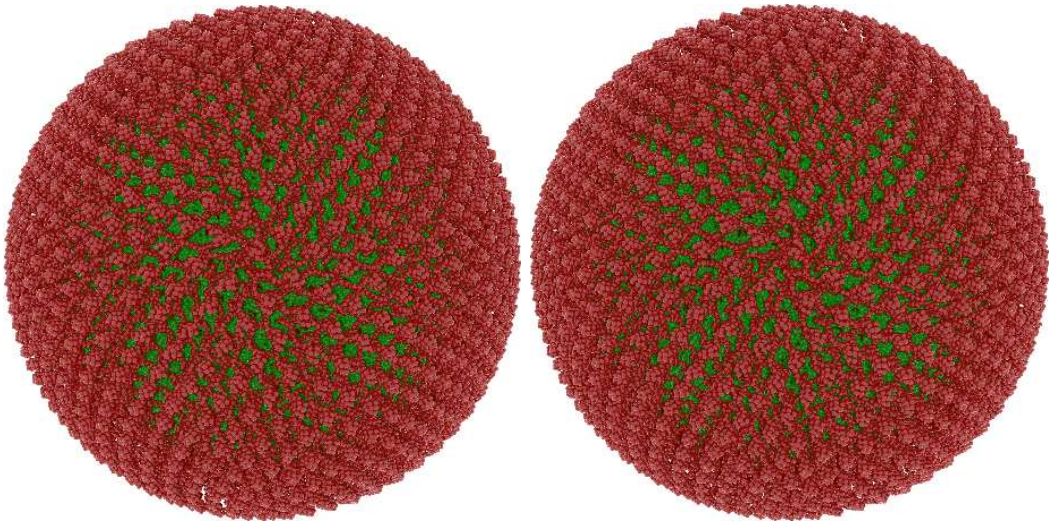


Figure B.3: A PEO-PEE vesicle formed using the *dpdtimestep* initial state creation tool.

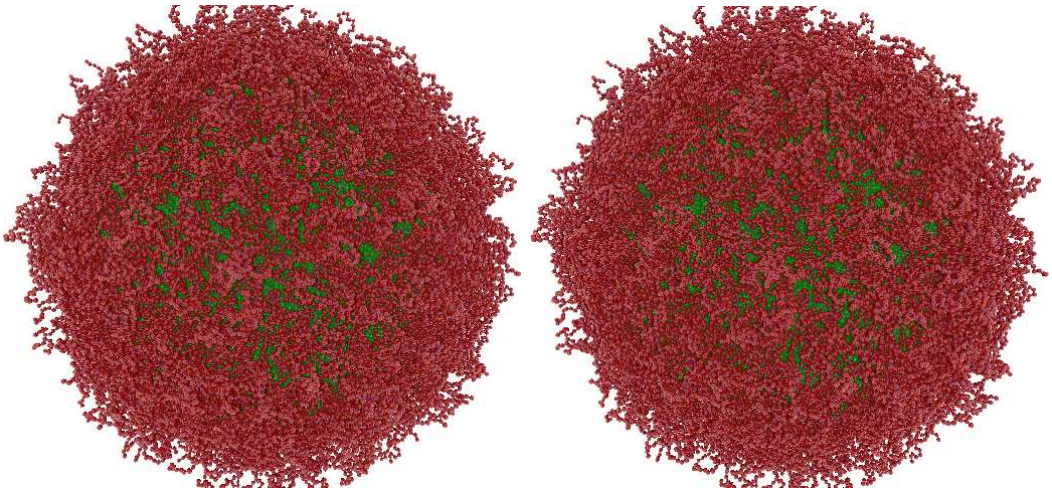


Figure B.4: The same PEO-PEE vesicle after simulation for 5000 time units.