The University of York Department of Computer Science

Submitted in part fulfilment for the degree of MEng.

Parallel Programming Tools for Exploring Immune System Development

Oliver Binns

30th April 2018

Supervisor: Dr. Fiona Polack, Dr. Kieran Alden

Number of words = 8908, as counted by texcount.
This includes the body of the report only.

Abstract

More powerful computers are paving the way for sophisticated simulations of complex biological systems which are inaccessible to live in-vitro study. Agent-Based Modelling has been previously used to implement this type of simulation[1]–[4], showing how the behaviour of a number of individual agents can contribute to the system as a whole.

The desire to create larger and more elaborate simulations, while still ensuring results can be obtained in a reasonable amount of time, makes it necessary to use parallel and distributed systems. Creating programs and simulations that make efficient use of this parallelism can be technically challenging. In particular it requires implementing an optimal distribution of tasks over the system and ensuring the integrity of any memory that is shared across multiple threads. Some significant problems, including a computer science skills shortage, are slowing down the rate of progress in this area.

This project investigates the challenges and advantages of using a parallel agent-based modelling platform, FLAME GPU, by re-implementing an existing sequential simulation of the immune-system, developed using MASON. During the building of the simulation, I have discovered that it is not trivial to compare between the old and new implementations, due to differences in the frameworks used. For this reason, the final comparison will be done directly to the domain model itself.

Finally, in order to improve the mapping from the biological domain model to FLAME GPU simulation code, I propose a Model-Driven Engineering (MDE) approach to development. This approach reduces the parallel computing skill required to develop FLAME GPU implementations and improves traceability from the biological domain to the implementation.

Acknowledgements

I would like to thank my supervisors, Fiona Polack and Kieran Alden, for their support and guidance throughout this project.

I also express my thanks to Richard Paige for his pastoral support throughout this academic year, as well as his help in obtaining access to a suitable GPU machine.

Finally, I would also like to thank the FLAME GPU development team at the University of Sheffield for their help in utilising the latest and pre-released features of this framework.

Contents

List of Figures

Acronyms

ABM Agent-Based Modelling. 3, 13, 18, 19

CoSMoS Complex Systems Modelling and Simulation Infrastructure. 21

CPU Central Processing Unit. 18

CUDA Compute Unified Device Architecture. 18

EVL Epsilon Validation Language. 23, 31

FLAME Flexible Large Scale Agent Modelling Environment. 19

FLAME GPU FLAME for the GPU. 3, 5, 19–24, 30, 31

GPGPU General Purpose GPU programming. 11

GPU Graphics Processing Unit. 11, 18, 19, 21

LTi Lymphoid Tissue inducer cell. 22

LTin Lymphoid Tissue initiator cell. 22

LTo Lymphoid Tissue organiser cell. 22

MASON Multi-Agent Simulator Of Neighborhoods. 3, 19, 22, 24

MDE Model-Driven Engineering. 3, 17, 19

Glossary

FLAME GPU An extension of the FLAME framework which uses GPUs for parallelism. 3, 5, 19–24, 30, 31

LTin A moving cell that can bind to any static cell. Responds to chemokine expression. 22

LTi A moving cell that can bind to any static cell. Believed to be responsible for initiating Peyer's Patch development. 22

LTo A static cell. Divides on stable bind with an LTin and LTi cell. 22

MASON A Java framework for producing simulations using ABM. 3, 19, 22, 24

Host In GPGPU programming, the computer containing the GPU is referred to as the host. 22

in-silico Scientific tests conducted using computer simulation. 10

in-vitro Scientific tests conducted using real biological systems under laboratory conditions. 10, 11

Peyer's Patch Clusters of cells that form in the small intestine. 11, 21

1 Introduction

Agent-Based Modelling (ABM) has been successfully used to simulate biological systems and develop new scientific knowledge[1], [3]. While traditional mathematical-based models are generally much more efficient to compute and produce good results when exploring how biological factors effect populations as a whole, ABM is particularly useful when considering how individuals and their interactions produce emergent system behaviour. Mathematical-based models are limited in this aspect as they assume that each individual within the population is identical. ABM explicitly represents each individual, meaning these can have state and attributes and are independent and fully autonomous. Unfortunately, these agent-based simulations require significantly more compute resources, thus taking a long time to run, particularly as simulations become more and more complex.

Here, and throughout this report, the term complexity does not necessarily refer to simulations that contains several components that are difficult to engineer. Rather, we discuss simulations that are made up of lots of simple components that interact with each other and the environment in such a way that the system behaviour is so elaborate that it can only be predicted by running the simulation.

As these simulations often contain non-deterministic behaviour, numerous runs are required to ensure that results are statistically significant. Consequently, it is important to ensure that the simulations make efficient use of the available compute resources, in order to gather these results in a reasonable amount of time. The inability to obtain simulation results in a reasonable time is one of the most significant constraints upon the current use of agent-based simulations within biological research.

Another constraint affecting the adoption of ABM simulations is the ease with which they can be created. At present, domain experts with knowledge of a particular biological system produce domain models of the system. With the aid of software developers, these domain models are then converted into platform models. The domain experts verify that the platform models are reasonable abstraction of the system, and they are implemented in code by the software developers. This is the CoSMoS

process (Fig. 1.1).

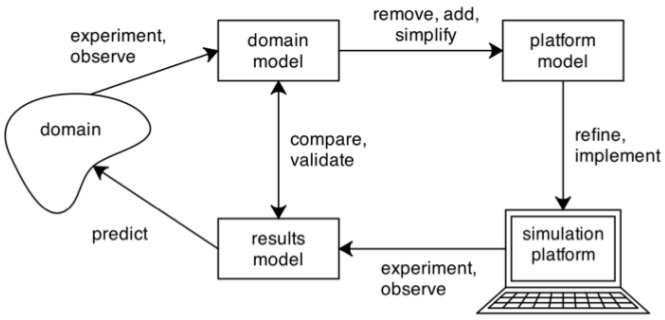


Figure 1.1: CoSMoS Process (Taken from [5])

A shortage of software developers who are proficient at creating efficient, parallel simulations is the current bottleneck in the adoption of the ABM simulations of processes. If this development process can be automated by creating general solutions for common biological processes, such as cell division, it will allow simulations to be created by domain experts.

1.1 Motivation

The motivation behind this project stems from the new knowledge that could be gained from the ability to easily create efficient, parallel biological simulations. As many biological systems cannot be fully studied in-vitro, simulations can provide a valid alternative. Indeed previous work has been used to produce novel biological hypotheses which have been shown to be statistically similar to those observed in the laboratory[1, p.174].

However, these previous simulations of biological systems have had to make significant trade-offs in their implementation. For example, 3D environments have had to be mapped into 2-dimensional simulations, cell migration into the system has been modelled as random rather than systematic[1], and environmental growth has been ignored[6]. The impact that these compromises have on the validity of the platform model as a representation of the domain is currently unknown. A core motivation of this project is that, by addressing the difficulties of creating efficient, parallel, agent-based simulations, we can support simulations with fewer trade-offs in their realisation.

There are several compelling reasons for the interest in simulation of immune-related biology. Firstly, extensive in-silico testing could be performed much more quickly than the current in-vitro testing, reducing the time it takes to research new drugs and cures. Drugs and cures would be available for patients earlier, saving many lives. Additionally, the significant financial burden of drug development, which is estimated to be around $2.9bn[7], could be reduced, freeing up heavily contested funds for additional research. Finally, the use of animal testing could be reduced and, in the long term, eliminated completely.

1.2 Project Aims

In summary, the aims of this project are to

1. Establish a firm grounding for the future development of new tools to allow fast, parallel simulations of biological systems to be easily created by non-technical users.
    a) Explore the findings of the new implementation and discuss how these can be generalised between different simulations.
    b) Discuss techniques for allowing non-technical users to easily create formal models that can be transformed into new simulation implementations.
2. Develop a **parallel** implementation of an existing **sequential** simulation of Peyer's Patch development and explore any speed increases that can be produced by using General Purpose GPU programming (GPGPU).
3. Review the use of simulations, with a particular focus on computational biology. This review explores the advantages that in-silico testing provides over in-vitro experimentation and the problems that must be overcome for the mass adoption of biological simulations.

1.3 Statement of Ethics

This project was conducted in accordance with the University of York's code of practice on ethics. This project does not involve human participants, so guidelines on informed consent and confidentiality will be

met. No confidential medical data or personal information has been used during the course of the project development. This project has involved no direct animal participation.

The ability to reduce, and ultimately replace, experimentation on animals is a positive ethical advantage of work on efficient immune-related simulation.

The simulation of the biological model is for the purpose of developing understanding of applying GPGPU methods to an agent based model of a biological system. It will not be used its current form to publish novel biological findings and does not fully simulate a biological process.

1.4 Report Structure

This report details the work done throughout the project and

- Chapter 2 gives an overview of simulations and the benefits and limitations of their use particularly with regard to computational biology.
- Chapter 3 states the tools used to facilitate the implementation of the new FLAME GPU simulation and presents details of model that was simulated.
- Chapter 4 details the development of an improved, inherently parallel, implementation of PPSim (*PPSim v2*) and the MDE tools that were developed to aid this work.
- Chapter 5 evaluates the progress that this project has made towards making **fast**, parallel simulations more **accessible**.

2 Literature Review

2.1 Simulation Background

Simulations are model-based imitations of a system which feature its key characteristics and behaviours. The system being modelled is referred to as the domain, while this simulation is known as the platform. Computer simulations are used in a wide range of disciplines on applications such as video games, medicine, product development and even nuclear weapons.

The models of systems used in simulations may have a varying amounts of abstraction. Simulations used for teaching will likely have models which remove significant amounts of complexity from the system. Using abstraction to remove some of the complexity present in the domain can also help to reduce the time it takes to run the simulation, a key focus of this project. Simulations used for video games tend to be as realistic as possible as realism has been shown to produce a higher level of immersion[8], a highly desirable attribute of games.

In particular this project will focus on simulations created using Agent-Based Modelling (ABM). ABM is a technique that explores the autonomous behaviour and interaction between a number of individuals, known as agents. The aim of this is to show how the individuals' behaviours interact to produce the system as a whole. This is in contrast to more traditional mathematical top-down models which consist of a set of equations which establish relationships between a set of variables.

This chapter begins by outlining the benefits and current limitations of using simulations *generally*. It then discusses some potential solutions to some of these limitations before applying this to some of the existing simulations that have been developed using ABM.

2.1.1 Benefits

Feasibility

Exploring computer simulations is can be far more feasible than exploring a real world environment. Video games are simulations which may allow

players to experience scenarios that they may not otherwise get the opportunity to encounter. For example, car racing games are significantly cheaper and safer than real life racing.

Real-world scientific testing may not be feasible for a number of reasons. Simulating the aerodynamics of new car designs virtually is far quicker and cheaper than creating multiple different prototypes for physical tests. Morality may be a factor, a good example of this is animal testing for cosmetic products or medicine. With nuclear weapons, legality too is a key issue, as some weapon testing is banned under a number of global treaties[9], [10]. Finally, some real-world tests may be too dangerous to perform, such as in the case of invasive medical examinations. In all of these examples, computer simulation is routinely used to reduce or replace real-world testing.

Reducing Complexity

Reducing complexity through abstraction allows better understanding of the system to be gained as the complexity may initially be overwhelming. Abstractions need to be made to an appropriate layer, models should represent the layer being investigated, but remove any unnecessary further detail. For example, attempting to model simulations exploring cell development down to the atomic or quantum level or up to an entire organism would be completely unnecessary. Attempting to do so with current technology would also be completely untenable. This is an important balancing act as, if too much detail is abstracted away, simulations will likely fail to produce the expected emergent behaviour[4].

Environmental Control

Computer simulations allow for the environment to be more easily controlled. While the initial setup of a realistic environment may be difficult, once this has been achieved the simulations can be run in a known, constant environmental context. It can be more easily ensured that the environment remains unaffected by any external factors. The ability to adjust external factors and independent variables that may affect the system on demand can also be particularly useful. This ability can be used for illustrating how different variables affect system processes.

Furthermore, simulations allow for additional tests to be easily added at a later date. For example, if measurements are not made for a particular variable, it can be easily added to the implementation and the simulation

can be easily re-run. This is not the case with in-vivo experimentation, which would require the laboratory to be fully set-up again, with new test-subjects for repeat experiments.

Visualisation

One of the biggest benefits of simulation is the ability to graphically visualise a system or its constituent parts. Using simulation for visualisation has number of benefits over attempting to demonstrate real world systems. Several of these benefits translate from the previous two sections- they can be more feasible to explore than a real world environment and featuring a reduced complexity can allow the key concepts to be understood without overwhelming the user.

Simulation can be particularly useful for visualising concepts for education. Particularly immersive educational visualisations can also be produced using virtual and augmented reality. The Virtuali-Tee is an educational tool that uses simulation and augmented reality to provide a view at the body's internal organs[11]. Little Journey aims to reduce kids' anxiety about surgery by providing a realistic tour of their hospital ward given by animated cartoon characters[12].

2.1.2 Limitations and Constraints

While simulations seem very useful across a wide number of fields, there are some significant limitations as to where and how they can be used.

Insufficient Domain Knowledge

A simulation is based on a model of a system. A model is an abstraction from reality representing only the necessary key characteristics and behaviours of the system. In order for the model to the be faithful to its original domain it must be able to produce demonstrably similar behaviour from similar system components.

A lack of knowledge regarding the domain of the simulation is one of the most significant constraints regarding its implementation. If this is the case, the model produced may be incorrect or abstractions may remove necessary detail required for the system to function as expected. With biological simulations in particular, it is often hard to make assumptions about how particular processes occur[4]. Unfaithful models can also be a result of *incorrect* knowledge of the domain. For complex systems, having

too many abstractions from the original domain may also invalidate the model and produce incorrect results[13, p.8], [4].

While knowledge of a system needs to be sufficient to produce reasonable assumptions, it need not be perfect. This is particularly the case with the type of simulation discussed later in this report. In this case, assumptions about the domain are made

Compute Power

Complex models with too few abstractions from their domain may require significant computing power to simulate. Additional abstractions may not be possible as they may invalidate the model[4]. In these cases, cutting edge hardware may be needed for the simulation to be run in an acceptable time.

Powerful hardware is expensive to access, so this may be a significant constraint on the ability to simulate.

Skills Shortage

The previous section discussed reliance that complex simulations have on the latest advanced hardware capabilities. However, advanced hardware alone will not necessarily allow a simulation to compute in a reasonable amount of time. With modern architectures which are becoming increasingly parallel, the simulation code must be tailored to take advantage of the computing power available. Efficient parallel programs that do this rely on the availability of experienced programmers. Many recent studies have highlighted the existence of significant computer science skills shortages, across the world[14], [15]. These skills shortages may be significantly limiting the possibility for cross-disciplinary work to utilise fast, advanced simulations.

Many initiatives are being rolled out to address this issue. Computing was added to the National Curriculum in 2013[16], [17] and numerous technology companies are also creating their own campaigns[18], [19]. It is likely that these will take a number of years to yield results, but should eventually mean that even domain-experts have a basic grasp of code.

Bugs

As with any form of computer program, mistakes can be made causing bugs to be present in the simulation code. Bugs may cause the simula-

tion to be incorrect meaning any hypotheses and results are based on incorrect data. This is linked to, but not the same as, having insufficient domain knowledge. Both of these limitations will cause the simulations to fail silently, produce incorrect results with no immediately obvious failure[CITE].

However, while these problems are specific to simulation, they are not dissimilar from the issues that can occur from poorly designed real-world testing.

If the simulation needs to be safety-critical, developing it using formal methods and refinement may be a good way to ensure that no bugs are introduced in the code.

2.2 Existing Simulations

2.2.1 Biological Simulations

Computational Biology is a relatively new field of study that has been growing significantly over the last decade[CITATION NEEDED]. Within Biology, simulation is often required as an alternative to invasive medical testing/animal testing.Simulation has even been proposed as a method for exploring a potential set of first principles and mathematics that are specific to biology which could even constitute a new subject- theoretical biology[20].

2.2.2 Agent-Based Modelling

As previously mentioned, ABM models the autonomous behaviour of individual agents and the interaction between them. Agents are often confused with objects, a commonly-used programming concept. However, it's important to emphasize the differences between them. In particular, while both agents and objects recognise the importance of interactions, objects are totally obedient to their method calls whereas agents have autonomy. This means that agents can interact and react to communications according to their own agenda.

While, Agent-Based Modelling is more computationally expensive than top-down mathematical modelling, it is also more natural to model and intuitive to parallelise[21]. Communication between individual agents can be difficult to implement in parallel but parallel communication is by no means limited to ABM.

Custom Code

A custom code ABM simulation was used to investigate the processes of auxin transport in plants.[4] The simulation focussed on how the transport canals in the stems of plants develop. Unfortunately this simulation did not produce the auxin transportation canals that would be expected from the biological model.

This report builds on an existing custom code example of ABM simulation which utilised Graphics Processing Unit (GPU) parallelism.[6] This simulation was built on a tried and tested domain model of Peyer's Patch development[1]. The simulation was successful in demonstrating the development of Peyer's Patches described in its domain model.

While it is clearly possible for agent-based simulations to be created from scratch there are several distinct disadvantages to doing so. Firstly, using existing frameworks avoid programmers having to reinvent the wheel, massively reducing the amount of code that they need to produce. As well as saving a significant amount of time, commonly used frameworks are often well tested making software bugs less likely. Furthermore, unlike simulations that are created using ABM libraries, they do not inherit any support for new tools and hardware that come from updates to the library. Custom code simulations must each be updated separately with any enhancements required for.

Proprietary Solutions

Some proprietary solutions are available to aid the creation of efficient ABM simulations. One example of this, with a reasonable level of industry support, is Biocellion[22]. While it is freely-available for non-profit use, it does not help with our aim of creating a simulation platform that is simple-to-use. While it makes the task of creating efficient parallel simulations which are portable far simpler, it requires a level of proficiency with the Linux OS, C++ programming and familiarity with mathematical modelling concepts. In particular, we cannot expect domain experts to be proficient at C++ programming. Software engineers will still be required in the simulation development process. In the future productivity layers which allow simulations to be created by less technical users will be added to Biocellion, but these are not yet available.[22] Biocellion's closed source approach rules out the ability for us to build on this work, while a license agreement has hindered adoption[23]

MASON

MASON is a multi-agent simulation library for Java. MASON has been previously evaluated as the one of the fastest ABM libraries available[24], however this evaluation took place over a decade ago and does not factor in any of the recent work into GPU parallelism[21]–[23]. Meanwhile, MASON only supports Central Processing Unit (CPU) parallelism, this will be discussed in more detail in Section 2.3.2.

FLAME GPU

The Flexible Large Scale Agent Modelling Environment (FLAME) project is an attempt to make GPU-based ABM simulations more accessible. FLAME simulations consist of two artefacts, an 'XML Model File' containing the structure of agents and their channels of communication and C source code file of 'Scripted Behaviour' which defines the actual behaviour of agents. FLAME for the GPU (FLAME GPU) is an extension of the original version of FLAME, where the simulations that are created are compiled down to parallelised CUDA code. This means the simulation can take full advantage of the power of modern NVIDIA GPUs.

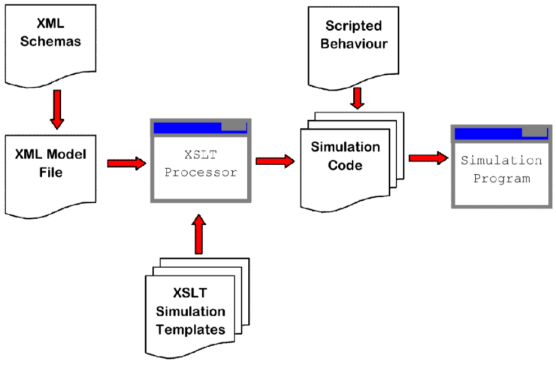


Figure 2.1: FLAME GPU Development Process (Taken from [21])

FLAME's use of MDE could gives domain experts with limited programming knowledge the ability to understand and help design the simulation models, however a software engineer is still needed for the final implementation.

While relying on a framework such as FLAME GPU can cause problems if support is stopped, this should be slightly less of a concern as FLAME GPU is an open source project. A more pressing concern with

relying on any framework is that implementations are restricted by any of the framework's limitations. For example, the current stable release of FLAME GPU does not allow new agents to be created by the Host between simulation steps.

However, the FLAME GPU framework does bring a number of advantages. Firstly, it provides a Message Passing Interface (MPI) which abstracts away from the underlying agent communication. This MPI supports spatial partitioning allowing messages to be passed only to nearby agents. Communication between threads which is one of the most difficult tasks in parallel programming. FLAME GPU manages this communication in its entirety, meaning programmers can instead focus on ensuring the agent behaviour is correct.

FLAME GPU also provides visualisations for simulations straight out of the box. The benefits of visualisation are discussed in Section 2.1.1. Visualisations in FLAME GPU run in real-time and can be useful for quickly verifying whether a simulation is behaving as expected.

Many other advantages come directly from FLAME GPU's use of MDE. FLAME GPU is continuously updated to support new hardware advances, such as multicore GPU architectures[21]. FLAME GPU simulations can always be certain of taking advantage of the latest hardware optimisations and portability features that the framework provides. Existing simulation models that are implemented on top of FLAME GPU may receive these performance improvements without any major code changes.

Some notable benefits of MDE are not exploited in the current implementation of FLAME GPU.

2.3 Improving Simulations

Significant constraints on when simulations can be used were discussed in Section 2.1. In order for simulations to become more widely adopted, some of these constraints must be overcome. In particular, this project focuses on two particular sections of which I believe will have the biggest immediate impact. This section discusses the potential solutions to each of these problems.

2.3.1 Ease of Creation

A major constraint on the adoption of simulations is how difficult and time-consuming they are to create. This constraint is exacerbated by

a current shortage of experienced programmers. Initial work to help overcome this skills shortage will likely come by providing tools that can automate work done undertaken by technical users in order to increase their productivity. Eventually, if this technical work can be fully automated, non-technical users will be able to easily create advanced simulations.

Model-Driven Engineering

This project has already discussed abstractions with regards to modelling for simulations. Abstractions are also particularly useful in software engineering. As software gets more complex, additional abstraction is generally needed in order to extract the important details of the implementation[25, p.24].

MDE is a software abstraction for creating and exploiting domain models, such as those that have been previously discussed with regards to simulations. MDE has shown a promising increase in understanding between stakeholders and can produce productivity gains when models are reused across projects[26].

Epsilon Epsilon is a set of tools and languages for supporting MDE. Epsilon provides both the standardised modelling and metamodelling languages and model management technologies that are required for successful MDE projects[27]. It provides tools for creating metamodels for new domain specific languages. Epsilon facilitates model validation, model-to-model transformation and code generation to be performed on for models conforming to this metamodel.[28]

Flexible Modelling Flexible Modelling tools could be a good method for allowing new simulations to be created more easily. Using flexible modelling, non-technical users would be able to create sketches which can be automatically processed by tools into formal models and prototype metamodels[29]. FlexiSketch is a good example of this and provides a good tool for creating models and metamodels for software development[30].

2.3.2 Speed Up

Simulations need to be able to generate results in a realistic amount of time. Many biological simulations include significant elements of

randomness meaning must be run a large number of times in order to produce statistically significant results[31]. With novel applications attempting to simulate increasingly complex models, it is becoming more and more difficult to ensure that they have reasonable runtimes. This section evaluates recent work into achieving faster simulations.

Machine Learning

An upcoming paper explores the possible use of machine learning to emulate and predict the output of an existing simulator[32]. This emulation method produced some very accurate results within seconds, however it still requires an initial training set, and so does not fully remove the desire for fast simulations. As the number of parameters increase, the training set size must also increase in order to produce accurate results. This technique also requires expertise in machine learning which conflicts with our other aim- to make simulations more accessible.

Parallelism

Parallelism fundamentally changes the game and allows computers to keep following Moore's law even as engineers are struggling to make transistors ever smaller and smaller[33]. As modern computers tend further towards parallelism to keep providing the speed-ups that have been inherent in the industry over recent years, new parallel algorithms need developing in order to take full advantage of the computing power available.

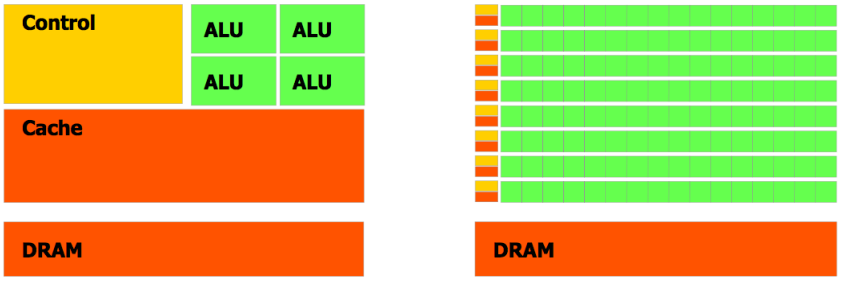
Managing parallel implementations of algorithms has significant overheads due to the need to ensure memory integrity when transferring data between parallel tasks. The different approaches to parallelism make different trade-offs, meaning some may be more suited to a particularly task than others. In order to ensure that the gains made by parallelising the task are not overshadowed by the overheads of managing a parallel implementation, the simulation programmer must have sufficient knowledge of the approach being used.

Distributed Systems A distributed system is a method for running a computer program across multiple computers on a network. Unlike parallelism that is local to a particular computer, with a distributed system there is generally no global clock to ensure that processors are kept in sync. In order for any distributed system model to provide speed

improvements, the gains provided by parallelising the data processing step must be substantial enough to overcome the time taken for data transfer between computers.

DMASON is an enhanced version of MASON library which aims to harness unused PCs as a makeshift distributed system. DMASON can also be set up to run on a compute cluster.

CPU vs GPU Modern computers provide two main means for parallelising code. We are building on a previous project[6] which laid the groundwork for this. This previous project outlined the choice between CPU and GPU parallelism and makes the case for exploring GPUs- simplisitically put, this is due to the significantly greater speed ups that can be achieved. Indeed it has been shown that GPU simulation on a standard desktop computer can easily produce performance rates that are better than those of a high-performance computing cluster[21]. One case study even reduced the time taken for a simulation to be computed from hours on a CPU to just seconds on a GPU[3].



(a) CPU (b) GPU

Figure 2.2: Silicon Space Allocation (Taken from [34, p.2])

The key difference between CPU and GPU parallelism lies in the Flynn's taxonomy classification[35] of architecture used by each. CPUs use a Multiple Instruction, Multiple Data (MIMD) paradigm- this is parallelism of instruction as different CPU cores can each be running different instructions concurrently. Meanwhile, GPUs use a Single Instruction, Multiple Data (SIMD) paradigm- this is parallelism of data as the same program is executed on a large quantity of data. These differences can be most starkly visualised by observing the difference in the allocation of transistors between the two components (Fig. 2.2). While the CPU devotes a large amount of space to data caching and flow control, the

GPU allocates the vast majority of its capacity to data processing.[34, p.2]

Multiple CPU cores are now commonplace in modern computers, including smartphones. In contrast to this, dedicated GPUs are normally only found in gaming PCs and consoles.

CUDA vs OpenCL A previous project compared CUDA and OpenCL in detail. The key advantage that OpenCL has over CUDA is its portability. Where CUDA is proprietary and runs only on NVIDIA hardware, OpenCL implementations are available for the majority of GPU hardware including AMD, Apple[36], Imagination and NVIDIA.[37] Nevertheless, this portability comes with a cost, code for different hardware must be tailored specifically to each hardware platform and performance has been shown to be consistently poorer[38]. The previous report also concluded that Compute Unified Device Architecture (CUDA) has better library support and is simpler to use[6].

3 Tools and Platform

3.1 Tools

This project was developed on a machine running Ubuntu 16.04 with Intel i5 (650 @ 3.20 GHz) and 8GB of RAM. The GPU utilised was an NVIDIA GeForce GTX 1050. GPU programming utilised the latest version of CUDA (9.1). CUDA was selected over OpenCL due to it being a requirement of FLAME GPU. Minimal knowledge of CUDA was needed for the development of the project as FLAME GPU handles the majority of the GPU hardware management.

In order to produce a working simulation of Peyer's Patch formation, I have used an unreleased version (v1.5) of FLAME GPU. Amongst other features, FLAME GPU v1.5 adds the ability to create agents from a host function between simulation steps. As FLAME GPU is developed openly, this version is freely available online[39]. FLAME GPU has been added to the project repository as a submodule, to ensure version compatibility between FLAME GPU and *PPSim v2*.

Eclipse Neon v4.6 and Epsilon v1.4 have been used with Java Runtime Environment 1.8.0 for developing the input XML model for FLAME GPU.

Version-control for both the simulation code and this report has been managed through Git and is available on GitHub. As this is an individual project, Git has been an acceptable solution. If the scope of the project is to expand, it may become necessary to move the modelling section of the development into a more suitable version-control system, such as EMFStore[40] which is specifically designed for models.

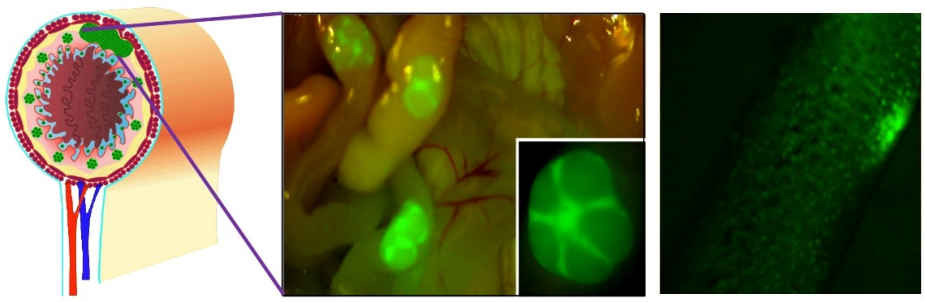
3.2 Peyer's Patch Case Study

This project focuses on simulation as a tool for exploring biological systems at cell level. It uses an existing sequential MASON simulation of Peyer's Patch development[1], [2], [31], [32], [41] (*PPSim*) and attempts to use parallel computer architectures in order to speed this simulation up. The sequential *PPSim* had a runtime of `94.265` seconds and `585,000`

executions were required for the statistical sampling technique being used. Even with a significant amount of CPU parallelism (138 processors) available, the combined runtime of these executions is inconvenient and would be intractable for larger scale simulations.[32] As previously stated, FLAME GPU can easily produce results to better a high-performance cluster, such as the one used to compute *PPSim*, even on a standard desktop PC.

3.2.1 The Domain Model

This domain model for *PPSim* was developed according to the CoSMoS process[13], which was introduced in Section 1. The model describes the cell behaviour over a 72 hour period in which Peyer's Patches develop. Peyer's Patches are clusters of cells that form on the lining of the small intestine (Fig 3.1).



(a) Adult Peyer's Patch (b) Foetal Peyer's Patch

Figure 3.1: Images of In-Vitro Peyer's Patch (Provided by Mark Coles, University of York)

The environment of the model is a 3-dimensional tube modelled in 2-dimensions. Agents leaving the environment in the x-direction are removed from the population and no longer considered. Agents leaving the environment in the y-direction wrap around to the origin.

In this model there are three types of agent which are all types of cells; LTo, LTin and LTi. State diagrams that describe the behaviour of each of these cells were presented as part of the original domain (Fig. 6.2) and platform models (Fig. 6.4)[1].

The model starts when LTi and LTin cells begin migrating into the environment. As such, at the beginning of the development process, neither of these cells are present. LTi and LTin cells migrate into the

environment throughout 72 hour time period being modelled. Initially these LTi and LTin cells move randomly around the environment.

Stationary LTo cells are randomly distributed around the environment. These cells are present at the start of the development, but are dormant until an LTin cell collides with them. When this collision occurs, the LTin may bind to the LTo cell and the LTo cell begins secreting a chemokine signalling protein. Upon each addition LTin collision, the LTo starts to release chemokine at a greater rate. Once the LTo move into this new state, they will also divide every 12 hours, resulting in new LTo cells.

LTi cells respond to the chemokine when the level in their local environment is great enough. When this happens, they move towards the LTo cell and eventually collide. Upon collision LTin cells may bind to the LTo cell, as determined by the latter's adhesion level. Neither LTin and LTi binds are permanent as the LTo's adhesion level may not be high enough to ensure prolonged contact. Each successive contact with an LTin cell increases the LTo's adhesion level, meaning that prolonged contact eventually becomes inevitable and the Peyer's Patch is formed.

3.3 The Platform Model

The platform model (Figs. 6.4 & 6.5) refines the domain model and states how unknown or non-deterministic biological behaviour can be modelled in-silico. In particular, the platform model specifies models for adhesion between cells and how LTo cells react to chemokine levels in their local environment[1], [2]. One notable *unknown* domain value is the rate that LTi and LTin cells y migrate into the environment. In the Platform Model this is modelled at a constant rate, such that the *known* correct number of cells is reached at the 12 hour time-step.

The platform model will be implemented using the FLAME GPU framework. This framework will be responsible for managing all data-structures and managing GPU memory across different threads containing agent instances. Variables and constants that pertain to the environment will be stored using FLAME GPU's implementation of global variables declared in the model file. Variables that concern specific agents only will be stored as variables under these agents. Agent variables will be kept to a minimum with distinct states being used to separate different agent behaviours as far as possible.

4 Methods

The chapter details the simulation experiment that has been performed. The experiment involved the development of a new GPU-based *PPSim v2* implementation as well as a new tool for mapping and transforming biological platform models into FLAME GPU simulations. Having previously established the current difficulties with biological simulation, this chapter also details how these are overcome in the implementation of this project.

4.1 FLAME GPU Simulation

As previously mentioned a FLAME GPU implementation requires the creation of two artefacts. A structural 'XML Model File' which defines the agents in the system and their allowed channels for communication. A C source code file of 'Scripted Behaviour' defines the actual behaviour of agents.

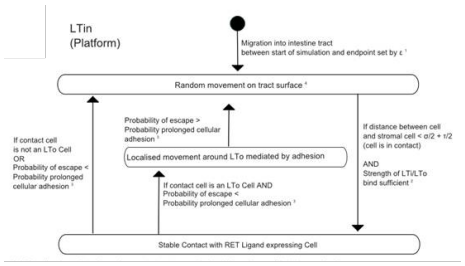
4.1.1 Structural Model

FLAME GPU defines agents as X-machines in its XML model. X-machines are structurally identical to finite state machines, other than that they have memory, which means it was possible to directly implement the platform model state diagrams directly into the FLAME GPU simulation (Fig. 4.1).

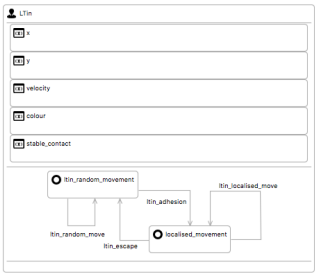
During the development of the model implementation, there were many occasions where the XML of the model because invalid because of human error. These mistakes either caused ill-formed XML or an XML model which did not conform the FLAME GPU model schema, which resulted in many tedious hours of debugging. In order to mitigate the issues caused by these mistakes, a new tool was created using the Epsilon Modelling Framework. The new tool ensures that any XML files that are created are valid XML and conform the the FLAME GPU model schema.

In the tool, FLAME GPU models can be designed using a graphical interface (Figs. 4.1b & 6.1) and then exported to the required XML

file format. Adding this GUI front end to FLAME GPU is a major step towards ensuring these parallel ABM simulations are more easily accessible by non-technical users.



(a) LTin Platform Model[1]



(b) LTin Agent in Graphical Editor

```
<gpu:xagent>
    <name>LTin</name>
    <description>Lymphoid Tissue initiator cell</description>
    <memory>
        <gpu:variable>
            <type>float</type>
            <name>x</name>
        ...
    </memory>
    <functions>
        <gpu:function>
            <name>ltin_random_move</name>
            <currentState>ltin_random_movement</currentState>
            <nextState>ltin_random_movement</nextState>
        ...
    </functions>
    <states>
        <gpu:state>
            <name>ltin_random_movement</name>
        </gpu:state>
        ...
        <initialState>ltin_random_movement</initialState>
    </states>
    <gpu:type>continuous</gpu:type>
    <gpu:bufferSize>5530</gpu:bufferSize>
</gpu:xagent>
```

(c) *Autogenerated* FLAME GPU XML Model for LTin Agent

Figure 4.1: Autogeneration of FLAME GPU Model File

In addition to ensuring that only valid models can be produced, the graphical tool has a number of other benefits. The original platform model (Fig. 4.1a) and the FLAME GPU implementation (Fig. 4.1b) can now be compared at a glance. For example, it is immediately clear that the platform model has been refined to include only two states in the final implementation. This is far less obvious when comparing the platform model (Fig. 4.1a) with the XML code (Fig. 4.1c). The removal of the 'Stable Contact' state is logical- this state is instantaneous and immediately transitions into either localised movement or random

movement, meaning its transitions can be merged into those which already exist between the other two states. The behaviour still exactly matches the platform model.

A number of other deviations were made from the platform model, mainly because of the time limitations of the project. The decision to only include a single central LTo cell in the simulation meant that the 'No Expression of RET-Ligand' state could be removed, as it cannot be reached.

Both the domain and platform model diagrams for LTi cells are missing a transition from the 'random movement' to 'contact' state. This is because LTi cells may collide with LTo cells, without having reacted to chemokine emission. This transition was present into the original MASON implementation of *PPSim*, but was never corrected in the state diagram. It has also been added into the new FLAME GPU implementation.

4.1.2 Behavioural Model

The agent behaviour is defined in a CUDA source code file. Code may be run on the Host between simulation steps, or on the Device during steps. Host functions have been used scarcely for minor administrative tasks, as they carry a data transfer penalty and do not benefit from GPU parallelism.

The only two tasks handled by the Host are tracking the current simulation time (by incrementing `SIM_STEP` between each step) and managing the migration of cells into the environment. The ability to create agents from a host function is currently only available in a pre-release version of FLAME GPU, which is why this version of the software was selected. Previously, agents could only be created during the simulation by other existing agents. Creating another agent type solely to manage LTo and LTin migration would needlessly complicate our platform model, which is highly undesirable. Furthermore, this implementation is very unintuitive and thus conflicts with our aim of simulations being simple to create.

LTo cells divide every 12 hours once they reach the 'Upregulation of Chemokines' state. A previous project on this topic stated that LTo cell division must be performed sequentially on a host function in order to prevent new cells occupying the same space[6]. However, by adding a force resolution step, as used in previous FLAME GPU simulations[3], this has been implemented in parallel, on the device. As well as the

speed up from parallelising the division step, implementing the feature in this way has also removed the cost of data transfer between the Device and Host.

Unfortunately, due to limitations of the FLAME GPU framework and the time constraints on the project, there are several features of the Platform Model which I have not been able to implement. Firstly, in this implementation the environment is static and does not grow. Environmental growth would not be technically challenging to implement in FLAME GPU. A Host function would increment the environment size variables at each time step, and transitions on each agent would allow them to adjust their positions accordingly the new size, in parallel. However, due to FLAME GPU's dependence on state machines, this transition would have to be present on every state for every agent, and thus would be very time-consuming to implement. On top of this, in the current version of FLAME GPU, these transitions, which would be identical for every state of each agent, would be classed as different functions, and thus each implemented sequentially. This would significantly increase the runtime of the simulation.

Secondly, once LTo cells transition into their 'mature' state, they no longer divide.

4.1.3 Execution

Initialisation

FLAME GPU simulations are initialised using another XML file specifying which, if any, agents are present at the beginning of the execution. In this platform model there is a single *active* LTo cell in the centre of the environment.

When in visualisation mode, the view is centred on the middle of the environment, with the environment bounds calculated from the position of the initial agents. In order to ensure that these bounds are calculated correctly, the simulation is initialised using an LTi cell placed at the maximum bounds of the environment (`7303, 254`). This single LTi cell will have minimal impact on the overall simulation behaviour.

Testing

As the new simulation is a re-engineering of the existing PPSim, it can inherit the validation of the existing simulation. This requires the

new simulation to be demonstrably similar with similar components, behaviours and results.

Talk about how the model was tested to ensure correctness

In order to obtain a fair representation of runtime, the simulation was executed as a console application `500` times, the same number of runs that was used for each parameter set of the MASON *PPSim* implementation. The simulation was run for its full length of `2880` steps (representing 72 hours).

4.2 Graphical Modelling Tool

In order to implement a tool to support my FLAME GPU development, the Epsilon Modelling Framework was selected. The decision to use this framework was simple, as it provides all the tools required and I was already familiar with it. The aim was for the tool to *directly* map the platform model structures into the required FLAME GPU XML model. This should significantly improve the model design experience and allow for domain experts to be kept more involved with the simulation creation. This new process is shown as part of the CoSMoS process in Fig. 4.2.

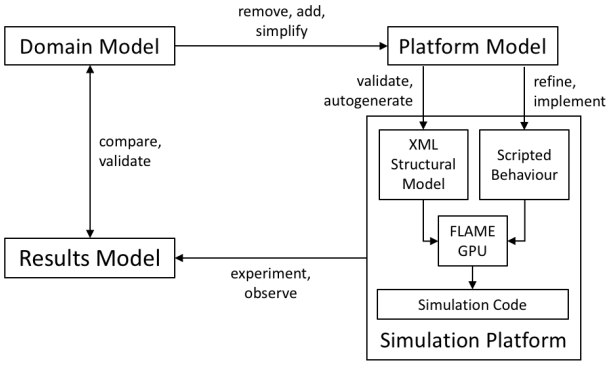


Figure 4.2: Autogeneration of FLAME GPU Artefact as part of the CoSMoS Development Process

Figure 6.1 shows the Peyer's Patch simulation created within this graphical editor. The XML model used for the final simulation was automatically generated from this diagram model.

4.2.1 Metamodel

The development of a metamodel was guided by the existing FLAME GPU XML model schema. This model schema is itself a metamodel for the FLAME GPU model file. The new metamodel was created using the Emfatic language, a convenience textual syntax for Ecore. This Emfatic text can be used to generate an Ecore model which, in turn, is used to generate the GMF Graphical Editor.

Much of the new metamodel matches the FLAME GPU schema. There is a top level simulation object which contains a list of agents, a list of messages that agents can send, etc. One notable difference here, is that as simulations contain exactly one environment, the data contained in this tag (constants, step functions, etc.) is stored at the simulation level, and no Environment class exists. The main difference in the new metamodel is its ability to utilise references to objects. Using references to objects ensures that the objects being referred to exist, which must be the case for model to be valid. For instance, we do not want to allow an agent to have an initial state that does not exist, setting this attribute as a reference in the metamodel ensures that this is not the case.

Due to the time constraints on this project, this metamodel is not quite a complete mapping to every FLAME GPU feature. Global Function Conditions, which act as a condition to determine whether a function should be applied to either all or none of the agents within a particular state, have not implemented. Additionally, global variable arrays cannot be created using the tool, these variables may only be of type int, float or double. Neither of these unimplemented features were required in the development of *PPSim v2*.

4.2.2 EuGENia

A GMF editor was created by adding EuGENia annotations into the Emfatic metamodel. These annotations are used to automatically generate the `.gmfgraph`, `.gmftool` and `.gmfmap` models required for the GMF Graphical Editor. EuGENiA provide a flexible range of options for displaying model data in different ways in the graphical editor[28]. Implementing the graphical editor required a number of important decisions here, in order for the tool to provide a logical user experience to modellers.

The majority of the graphical editor design was straightforward, with top level items such as agents, global variables and messages being

displayed as nodes on the diagram. Variables and states that belong to agents are displayed as inner nodes in containers within these nodes. EuGENiA allows transitions between states to be displayed as directional links, meaning the state diagrams for each agent can be displayed in the same way as they appear on the domain diagram (Fig. 4.1b).

A number of implementations for transition conditions were considered during development. Due to the limitations in the GMF Grapical Editor, values under containment cannot be edited unless they are displayed on the diagram. For obvious reasons, the transitions which are displayed as links, cannot display nodes on the diagram inside them, or link to them. In order to add edit the conditions on these transitions, the tree-based editor was used. This was recommended as the simplest approach in personal communication with Dimitris Kolovos, one of the members of the Epsilon team. Other options that were considered include implementing the diagram as a Petri net where conditions are displayed as intermediary nodes in between two states.

4.2.3 Epsilon Generation Language (EGL)

Epsilon Generation Language (EGL) was used to implement the transformation of the Epsilon GMF models into a FLAME GPU XML model. Since the tree structure of the GMF models already closely match the FLAME GPU model schema, this process was trivial and simply involved adding in the required XML tags in between the model data and retrieving the correct identifiers for any object references.

```
<states>
    [%for (state in agent.states){%]
    <gpu:state>
        <name>[%=state.name%]</name>
    </gpu:state>
    [%}%]
    <initialState>[%=agent.initialState.name%]</initialState>
</states>
```

Figure 4.3: Trivial Generation of FLAME GPU XML Model

4.2.4 Epsilon Validation Language (EVL)

The EVL has allowed well-formedness constraints to be implemented for the FLAME GPU input model. This should provide intuitive feedback (Fig. 4.4) in cases where the model may be incorrect, ensuring that all generated XML models conform to the FLAME GPU input model schema. Many of the EVL constraints have been used to cover shortcomings in

the Ecore metamodel. In particular, Ecore allows any attribute to be null, whereas in the majority of cases XML tags are not optional in FLAME GPU. Figs. 4.4 & 4.5 show how the EVL constraints would deal with a null initial state value for the LTin agent.

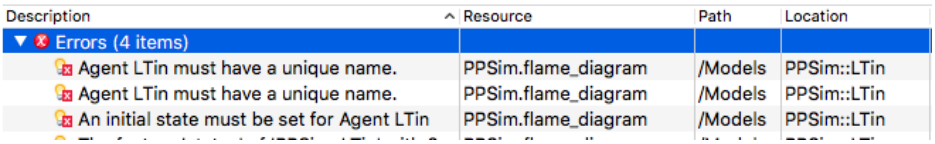


Figure 4.4: Graphical validation using EVL constraints

As well preventing null values, preventing obviously nonsensical values has also been a key use for EVL constraints. An example of this is ensuring that agents select one of their own states as their initial state, rather than one belonging to another agent. This will help to reduce the number of model errors that reach the FLAME GPU implementation stage.

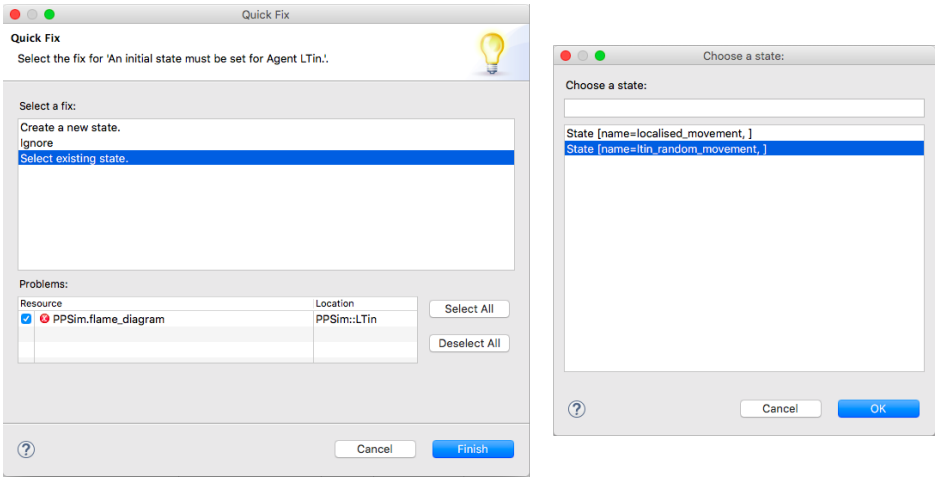


Figure 4.5: Quick fix of invalid model

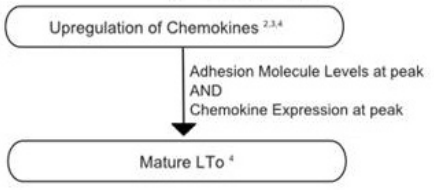
Quick fixes (Fig. 4.5) have also been implemented to guide the user through easily correcting their invalid model. The error feedback provided by these quick fixes is much more intuitive to non-technical users than that provided by FLAME GPU.

5 Results and Evaluation

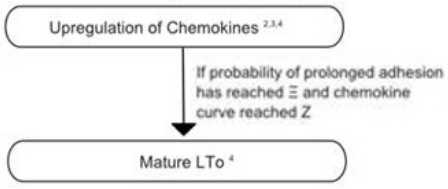
5.1 Findings

5.1.1 Ease of Creation

The new Epsilon tool has shown that direct mappings from a biological platform model to a simulation code are possible. Based on this, further work is now being developed with Paul Richmond's FLAME GPU group, to enhance the accessibility of FLAME GPU for simulation developers. This should allow the simulation developer to focus ensuring domain details are correct, rather than battling the agent platform. An improved version of this mapping, which abstracts away all FLAME GPU implementation details such as partitioning, could bring additional benefits in this area, including ensuring that the simulation code always supports the latest hardware features.



(a) Section of Domain Model



(b) Section of Platform Model

```
<gpu:function>
    <name>mature</name>
    <currentState>chemokine_upregulation</currentState>
    <nextState>matured</nextState>
    <condition>
        <lhs><condition>
            <lhs>
                <value>CHEMO_LOWER_ADJUST</value>
            </lhs>
            <operator>==</operator>
            <rhs>
                <agentVariable>linear_adjust</agentVariable>
            </rhs>
        </condition></lhs>
        <operator>&amp;amp;amp;</operator>
        <rhs><condition>
            <lhs>
                <value>MAX_ADHESION_PROBABILITY</value>
            </lhs>
            <operator>==</operator>
            <rhs>
                <agentVariable>adhesion_probability</agentVariable>
            </rhs>
        </condition></rhs>
    </condition>
    ...
```

(c) Fragment of FLAME GPU Model

Figure 5.1: Traceability of `Mature` transition from Domain Model to FLAME GPU Implementation

An additional benefit of this mapping, provides traceability from the domain model through to the simulation code. This can be clearly seen in Fig. 5.1, which shows how the transition from 'Upregulation of Chemokines' to 'Mature LTo' is refined from the Domain Model to Platform Model and then implemented in FLAME GPU. This traceability results in greater certainty that the simulation is faithful to the domain and no bugs have been produced during the implementation.

Furthermore, additional traceability helps to ensure that all artefacts contained in the domain model, platform model and simulation code are kept up-to-date with new information. In Section 4.1.1, a transition which was missing in the original domain and platform models was discussed. In the new process, where FLAME GPU model implementations are generated from the platform model, this transition could not be added into the implementation without first being added to the platform model.

This also tool aids development by finding and correcting model mistakes using EVL, this reduce the amount of time that developers spend debugging their code and increase productivity. Clearly this shows that the new process could help to detect mistakes in both the domain and platform models.

This work is ongoing.

5.1.2 Speed Up

Implementation Differences

Unfortunately a number of differences between the MASON and FLAME GPU platforms have meant that it is not trivial to compare them.

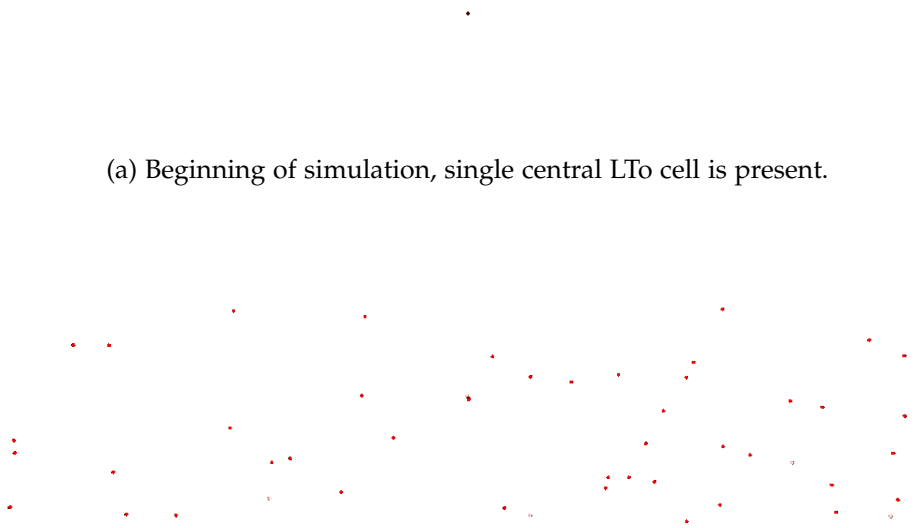
FLAME GPU's use of Message Passing to send adhesion probabilities in parallel has given rise to the possibility of outdated variables. In order to ensure maximum parallelism, the implementation used means that the probability of adhesion is the same for each collision that occurs in the same step of the simulation. If multiple LTi cells collide with an LTo cell in the same step, the behaviour will be somewhat different to that of the MASON implementation. Before the next step, the adhesion probability emitted by the LTo is updated for each collision, so the first collision of each step behaves the same as the MASON implementation. The impact of this is likely minimal as multiple collisions are unlikely to occur in the same timestep, due to the sparse population of the environment.

This initial hypothesis of the two platform models being comparable is

untenable.

Comparison

Instead, a new hypothesis is that the platform model for the FLAME GPU implementation is comparable to the original biological model. If our new platform model can be shown to be refined from the original PPSim domain model, we can say that this simulation also shows Peyer's Patch development, and thus is a faster representation of this biological process.

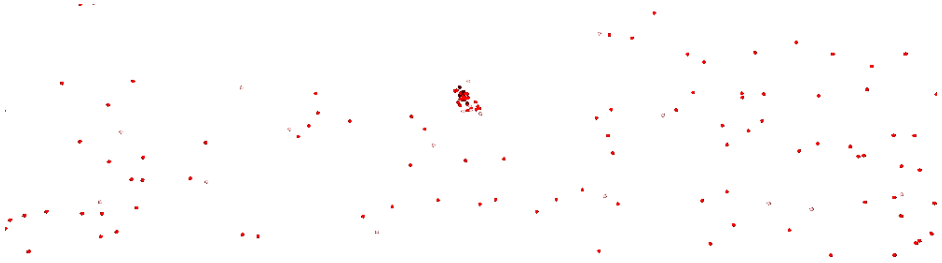


(a) Beginning of simulation, single central LTo cell is present.

(b) LTin & LTi cells migrate into the environment and activate LTo cell.

FLAME GPU provides a visualisation which can be used to demonstrate the appearance of the expected emergent behaviour. The expected behaviour of the simulation is that the Peyer's Patch cell clusters form around the initial LTo cell. This behaviour seems to be present in the *PPSim v2* simulation, as shown by Fig. 5.2, however this has not been verified by a domain expert.

The new version of the simulation has an average runtime of 19.352.



(c) Chemokine emission & LTo cell division causes Peyer's Patch to form.

Figure 5.2: Visualisation of *PPSim v2* produced using FLAME GPU

5.2 Conclusion

This project began with an ambitious set of goals, stated in Section 1.2. Over the course of the project, the use of simulations as well as its advantages over in-vitro experimentation has been reviewed. The biggest problems affecting the adoption of ABM simulations have been identified. Most significantly is that fast simulations of complex models are far too difficult to create, meaning experienced software developers are required. This has fulfilled our Aim 3.

The new implementation has shown that cell division *can be generalised between different simulations*. Other general features such as random cell movement has been discussed as another suitable candidate for generalisation. This has met and exceeded this part of the original aim.

MDE has been proposed as a *[technique] for allowing non-technical users to easily create formal models that can be transformed into new simulation implementations*. The MDE approach, taken in this project, has already realised numerous benefits in this area. The new graphical tool allows non-technical users to easily transform parts of *existing* platform models into FLAME GPU structural models. The reuse of artefacts is a fundamental part of MDE. This has given domain experts the ability to modify key simulation parameters, such as the sizes of cells and cell input rates. This ability is very useful for experimentation and for refining the simulation if new biological knowledge is discovered. The adjusted parameters can be automatically realised in the simulation code, something which would have previously required a software developer.

This is a *firm grounding for the future development of new tools* that will allow non-technical users to easily create fast, parallel simulations of biological systems.

An initial aim of this project was *'to explore the speed increases that can be produced by using GPGPU programming'*. A simulation has been created that displays the formation of Peyer's Patches, and while this has been shown to execute in less time that the original sequential *PPSim* there are too many differences in their implementations to directly compare them. The runtime of *PPSim v2* is still likely to be inconvenient if a great number of runs is needed. A sensible approach could be to use this faster simulation to generate training data for the machine learning approach described in [32]. Alternatively, a further level of parallelism, where different executions are run across multiple GPUs could be a way to ensure a more realistic total runtime.

In conclusion...

5.3 Further Work

5.3.1 Peyer's Patch Case Study

While the simulations are not entirely comparable, I have shown that fast, parallel implementations can be created using FLAME GPU. Further work on this topic is needed to show that these implementations can still produce results in an acceptable time when scaled up. With this simulation in particular some options for increasing the scale include modelling in 3-dimensions, adding further cell types and biological factors and cell types into the model, and simulating the whole length of the gut rather than a small section.

While the two Peyer's Patch simulations are both derived from the same domain model, there are clearly significant implementation differences between these, as is discussed earlier in this report. Future work will need to explore the meaning of the differences between simulations and whether that have any effect on their validity against the domain model.

Spartan is a tool for understanding relationships and providing novel biological insight into simulation behaviour. Spartan was use for analysing the results of the original PPSim implementation[31] and may provide an insight into validity of the parallel *PPSim v2* implementation.

5.3.2 FLAME GPU

A number of enhancements to the open-source FLAME GPU framework would have made the implementation of *PPSim v2* easier to create and faster to run.

Firstly, a method to implement global agent 'step' functions, that is to say, functions which are applied to every agent of a particular type, regardless of their current state. This would allow the environmental growth feature to be easily implemented, with a global step function used to update agent positions as the environment grows.

5.3.3 Software Generalisibility

While the use of Epsilon has allowed for domain experts to be kept involved with the model implementation, there is still a final, most technically challenging, stage of the simulation creation where the agent behaviour is programmed. Further work will need to study general agent behaviour in these forms of biological simulation. The reuse of a previous simulation's implementation of cell division behaviour, has shown that at least some behaviour is common across different simulations. The power of MDE is such that this repeated behaviour should be generalised to reduce the time needed and prevent the mistakes that occur during reimplementation of the similar software features.

On top of this, with the current implementation, technical platform-specific implementation details (layer functions and agent partitioning) have become part of the model. Ideally these should be extracted from the model implementation and automated.

5.3.4 Hardware Availability

One of the greatest challenges of this project has been gaining access to NVIDIA GPU hardware. While these are available off the shelf in most high-street computer retailers, they are not commonly found as part of standard desktop PCs, which generally containing integrated graphics hardware. Indeed, none of the software lab PCs at the University of York contain the dedicated graphics chips required to run this software. Future work could reevaluate whether the benefit of a cross-platform API, such as OpenCL, could outweigh the performance benefit provided by CUDA.

Bibliography

[1] K. J. Alden, '"Simulation and Statistical Techniques to Explore Lymphoid Tissue Organogenesis"', PhD thesis, University of York, Heslington, York, 2012.

[2] K. Alden, J. Timmis, P. S. Andrews, H. Veiga-Fernandes and M. C. Coles, '"Pairing experimentation and computational modeling to understand the role of tissue inducer cells in the development of lymphoid organs"', *Frontiers in immunology*, vol. 3, p. 172, 2012.

[3] P. Richmond, D. Walker, S. Coakley and D. Romano, '"High performance cellular level agent-based simulation with FLAME for the GPU"', *Briefings in Bioinformatics*, vol. 11, no. 3, pp. 334–347, 2010. DOI: 10.1093/bib/bbp073.

[4] P. Garnett, S. Stepney and O. Leyser, '"Towards an Executable Model of Auxin Transport Canalisation"', in *Proceedings of the 2008 Workshop on Complex Systems Modelling and Simulation, York, UK, September 2008*, 2008, pp. 63–91. [Online]. Available: http://www.cs.kent.ac.uk/pubs/2008/2920.

[5] M. N. Read, '"Statistical and Modelling Techniques to Build Confidence in the Investigation of Immunology through Agent-Based Simulation"', PhD thesis, University of York, Heslington, York, 2011.

[6] P. Drew, '"Parallel Programming Tools for Exploring Immune System Development"', Master's thesis, University of York, Heslington, York, 2017.

[7] J. A. DiMasi, H. G. Grabowski and R. W. Hansen, '"Innovation in the pharmaceutical industry: New estimates of R&D costs"', 2016, pp. 20–33. DOI: 10.1016/j.jhealeco.2016.01.012.

[8] K. Cheng and P. A. Cairns, '"Behaviour, Realism and Immersion in Games"', in *CHI '05 Extended Abstracts on Human Factors in Computing Systems*, ser. CHI EA '05, Portland, OR, USA: ACM, 2005, pp. 1272–1275, ISBN: 1-59593-002-7. DOI: 10.1145/1056808.1056894. [Online]. Available: http://doi.acm.org/10.1145/1056808.1056894.

[9] United Nations Office for Disarmament Affairs. (Oct. 1963). Partial Nuclear Test Ban Treaty, [Online]. Available: http://disarmament.un.org/treaties/t/test_ban (visited on 08/02/2018).

[10] U.S. Department of State. (Dec. 1990). Threshold Test Ban Treaty, [Online]. Available: https://www.state.gov/t/isn/5204.htm (visited on 08/02/2018).

[11] Curiscope Ltd. (). Curiscope, [Online]. Available: https://www.curiscope.co.uk (visited on 05/02/2018).

[12] Little Sparks Hospital Ltd. (). Little Sparks Hospital, [Online]. Available: https://littlesparkshospital.com (visited on 05/02/2018).

[13] P. S. Andrews, F. A. C. Polack *et al.*, '"The CoSMoS Process, Version 0.1: A Process for the Modelling and Simulation of Complex Systems"', 2010. [Online]. Available: http://www.cs.york.ac.uk/ftpdir/reports/2010/YCS/453/YCS-2010-453.pdf (visited on 19/04/2018).

[14] Ecorys UK, '"DIGITAL SKILLS for the UK ECONOMY"', Jan. 2016. [Online]. Available: https://www.gov.uk/government/publications/digital-skills-for-the-uk-economy (visited on 15/01/2016).

[15] Microsoft. (Dec. 2012). Investing in American Innovation and the Next Generation, [Online]. Available: https://blogs.microsoft.com/on-the-issues/2012/12/12/investing-in-american-innovation-and-the-next-generation/ (visited on 16/01/2018).

[16] Department for Education, UK. (11th Sep. 2013). National curriculum in England: computing programmes of study, [Online]. Available: https://www.gov.uk/government/publications/national-curriculum-in-england-computing-programmes-of-study (visited on 19/04/2018).

[17] ——, (3rd Jun. 2014). Microsoft, Google and IBM to help train computing teachers, [Online]. Available: https://www.gov.uk/government/news/microsoft-google-and-ibm-to-help-train-computing-teachers (visited on 23/04/2018).

[18] Apple. (24th May 2017). Apple launches app development curriculum for high school and community college students, [Online]. Available: https://www.apple.com/uk/education/teaching-code/ (visited on 19/04/2018).

[19] Microsoft News Center. (16th Sep. 2015). Microsoft expands global YouthSpark initiative to focus on computer science, [Online]. Available: https://news.microsoft.com/2015/09/16/microsoft-expands-global-youthspark-initiative-to-focus-on-computer-science/ (visited on 23/04/2018).

[20] D. Noble, '"The rise of computational biology"', *Nature Reviews Molecular Cell Biology*, vol. 3, pp. 459–463, 2002. DOI: 10.1038/nrm810.

[21] P. Richmond and D. Romano, '"Template-Driven Agent-Based Modeling and Simulation with {CUDA}"', in *{GPU} Computing Gems Emerald Edition*, ser. Applications of GPU Computing Series, W.-m. W. Hwu, Ed., Emerald Edition, Boston: Morgan Kaufmann, 2011, pp. 313–324, ISBN: 978-0-12-384988-5. DOI: https://doi.org/10.1016/B978-0-12-384988-5.00021-8.

[22] S. Kang, S. Kahan, J. McDermott, N. Flann and I. Shmulevich, '"Biocellion : accelerating computer simulation of multicellular biological system models"', *Bioinformatics*, vol. 30, no. 21, pp. 3101–3108, 2014. DOI: 10.1093/bioinformatics/btu498. [Online]. Available: http://dx.doi.org/10.1093/bioinformatics/btu498.

[23] A. Ghaffarizadeh, R. Heiland, S. H. Friedman, S. M. Mumenthaler and P. Macklin, '"PhysiCell: An open source physics-based cell simulator for 3-D multicellular systems"', *PLOS Computational Biology*, vol. 14, no. 2, pp. 1–31, Feb. 2018. DOI: 10.1371/journal.pcbi.1005991. [Online]. Available: https://doi.org/10.1371/journal.pcbi.1005991.

[24] S. F. Railsback, S. L. Lytinen and S. K. Jackson, '"Agent-based Simulation Platforms: Review and Development Recommendations"', *SIMULATION*, vol. 82, no. 9, pp. 609–623, 2006. DOI: 10.1177/0037549706073695.

[25] R. E. Bryant and D. R. O'Hallaron, *Computer Systems: A Programmer's Perspective*, 2nd ed. Prentice Hall, 2011.

[26] J. Hutchinson, J. Whittle and M. Rouncefield, '"Model-driven engineering practices in industry: Social, organizational and managerial factors that lead to success or failure"', *Science of Computer Programming*, vol. 89, no. Part B, pp. 144–161, 2014, Special issue on Success Stories in Model Driven Engineering, ISSN: 0167-6423. DOI: https://doi.org/10.1016/j.scico.2013.03.017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167642313000786.

[27] D. S. Kolovos, R. F. Paige and F. A. Polack, '"The epsilon object language (EOL)"', in *European Conference on Model Driven Architecture-Foundations and Applications*, Springer, 2006, pp. 128–142.

[28] D. Kolovos, L. Rose, A. García-Domínguez and R. Paige, *The Epsilon Book*. 30th Nov. 2017. [Online]. Available: https://www.eclipse.org/epsilon/doc/book/ (visited on 17/12/2017).

[29] R. F. Paige, A. Zolotas and D. Kolovos, '"The Changing Face of Model-Driven Engineering"', in *Present and Ulterior Software Engineering*, M. Mazzara and B. Meyer, Eds. Cham: Springer International Publishing, 2017, pp. 103–118. DOI: 10.1007/978-3-319-67425-4_7. [Online]. Available: https://doi.org/10.1007/978-3-319-67425-4_7.

[30] D. Wüest, N. Seyff and M. Glinz, '"FlexiSketch: a lightweight sketching and metamodeling approach for end-users"', *Software & Systems Modeling*, Sep. 2017, ISSN: 1619-1374. DOI: 10.1007/s10270-017-0623-8. [Online]. Available: https://doi.org/10.1007/s10270-017-0623-8.

[31] K. Alden, M. Read, J. Timmis, P. S. Andrews, H. Veiga-Fernandes and M. Coles, '"Spartan: A Comprehensive Tool for Understanding Uncertainty in Simulations of Biological Systems"', *PLOS Computational Biology*, vol. 9, no. 2, pp. 1–9, Feb. 2013. DOI: 10.1371/journal.pcbi.1002916. [Online]. Available: https://doi.org/10.1371/journal.pcbi.1002916.

[32] K. Alden, J. Cosgrove, M. Coles and J. Timmis, '"Using Emulation to Engineer and Understand Simulations of Biological Systems"', unpublished.

[33] H. Sutter. (Dec. 2004). The Free Lunch is Over: A Fundamental Turn Toward Concurrency in Software, [Online]. Available: http://www.gotw.ca/publications/concurrency-ddj.htm (visited on 08/02/2018).

[34] NVIDIA. (Feb. 2018). CUDA C Programming Guide, [Online]. Available: https://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf (visited on 25/04/2018).

[35] M. J. Flynn, '"Some computer organizations and their effectiveness"', *IEEE transactions on computers*, vol. 100, no. 9, pp. 948–960, 1972.

[36] Apple Inc. (8th Aug. 2013). About OpenCL for OS X, [Online]. Available: https://developer.apple.com/library/content/documentation/Performance/Conceptual/OpenCL_MacProgGuide/Introduction/Introduction.html (visited on 28/04/2018).

[37] The Khronos Group. (). Conformant Companies - The Khronos Group, [Online]. Available: https://www.khronos.org/conformance/adopters/conformant-companies#opencl (visited on 28/04/2018).

[38] K. Karimi, N. G. Dickson and F. Hamze, '"A Performance Comparison of CUDA and OpenCL"', *CoRR*, vol. abs/1005.2581, 2010. arXiv: 1005.2581. [Online]. Available: http://arxiv.org/abs/1005.2581.

[39] FLAMEGPU. (). FLAMEGPU, [Online]. Available: https://github.com/FLAMEGPU/FLAMEGPU/branches (visited on 18/03/2018).

[40] M. Koegel and J. Helming, '"EMFStore: a model repository for EMF models."', in *Proceedings - International Conference on Software Engineering*, vol. 2, Cape Town, South Africa, Jan. 2010, pp. 307–308. DOI: 10.1145/1810295.1810364.

[41] A. Patel, N. Harker, L. Moreira-Santos, M. Ferreira, K. Alden, J. Timmis, K. Foster, A. Garefalaki, P. Pachnis, P. Andrews *et al.*, '"Differential RET signaling pathways drive development of the enteric lymphoid and nervous systems"', *Sci. Signal.*, vol. 5, no. 235, ra55–ra55, 2012.

6 Appendix

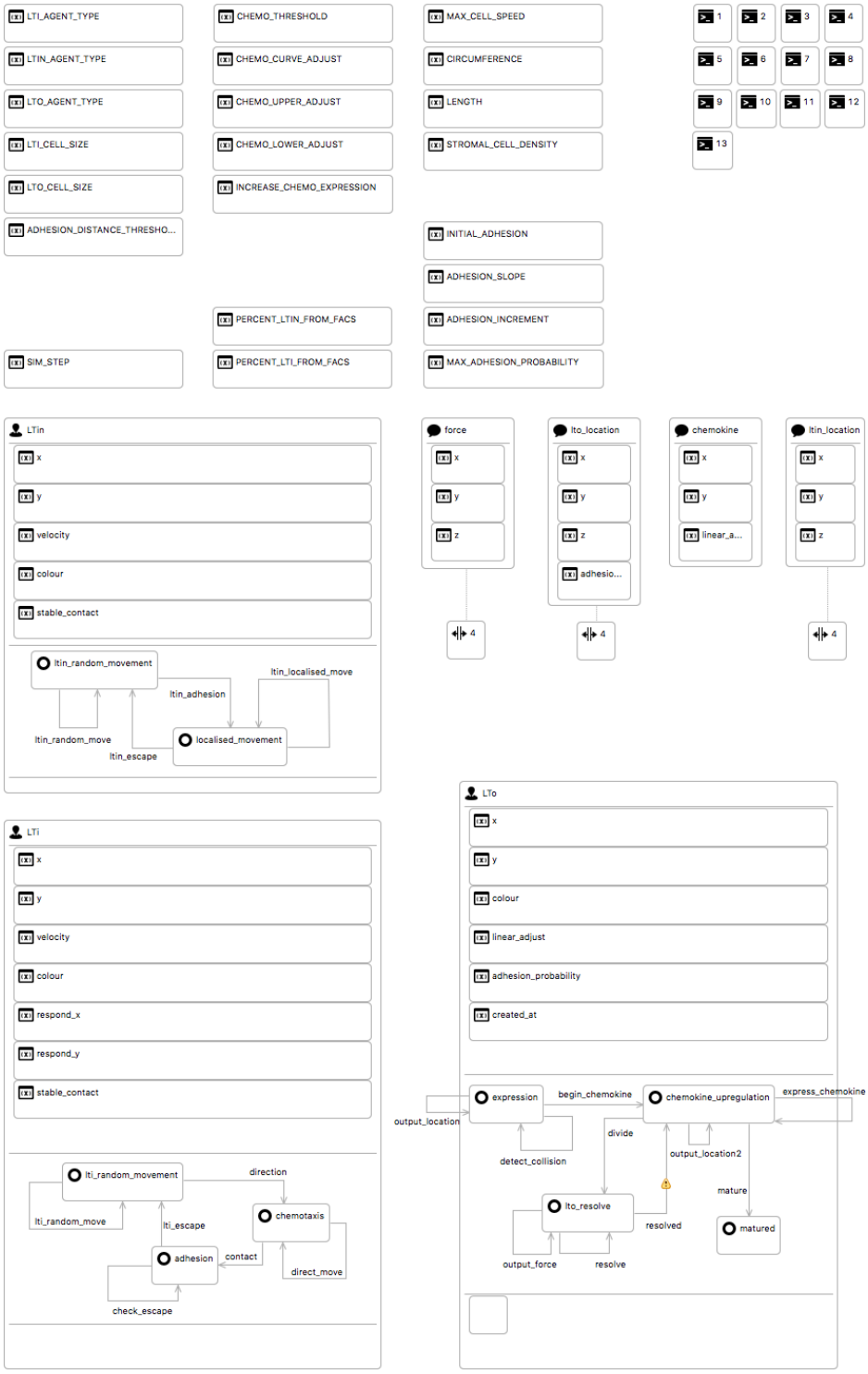
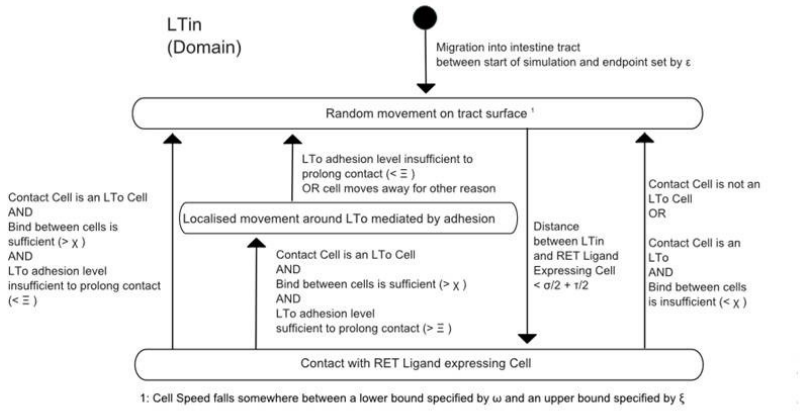
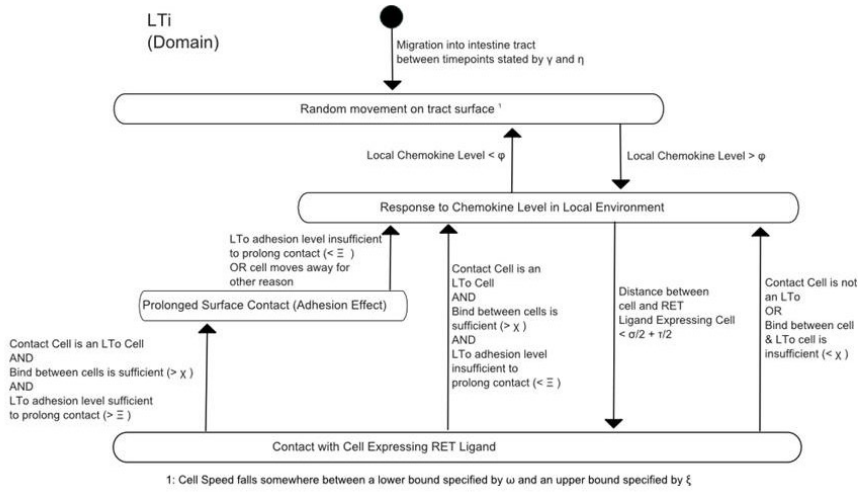


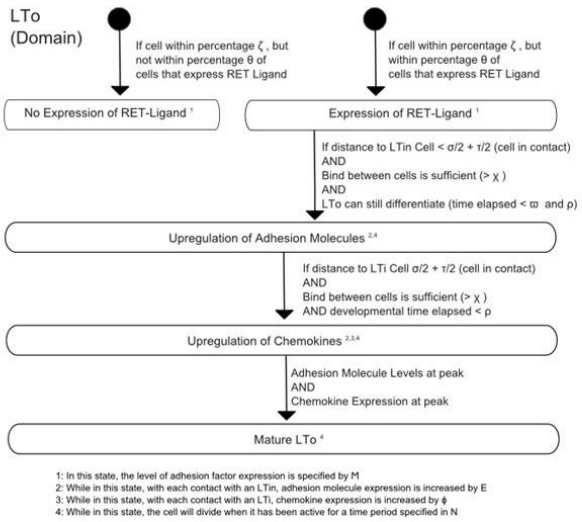
Figure 6.1: Graphically produced FLAME GPU Simulation model for Peyer's Patch


LTin
(Domain)
Migration into intestine tract
between start of simulation and endpoint set by ε
Random movement on tract surface 1
LTo adhesion level insufficient to
prolong contact (< Ξ)
OR cell moves away for other reason
Localised movement around LTo mediated by adhesion
Contact Cell is an LTo Cell
AND
Bind between cells is
sufficient (> χ)
AND
LTo adhesion level
insufficient to prolong contact
(< Ξ)
Contact Cell is an LTo Cell
AND
Bind between cells is sufficient (> χ)
AND
LTo adhesion level
sufficient to prolong contact (> Ξ)
Distance
between LTin
and RET Ligand
Expressing Cell
< σ/2 + τ/2
Contact Cell is not an
LTo Cell
OR
Contact Cell is an
LTo
AND
Bind between cells
is insufficient (< χ)
Contact with RET Ligand expressing Cell
1: Cell Speed falls somewhere between a lower bound specified by ω and an upper bound specified by ξ

(a) LTin


LTi
(Domain)
Migration into intestine tract
between timepoints stated by γ and η
Random movement on tract surface 1
Local Chemokine Level < φ
Local Chemokine Level > φ
Response to Chemokine Level in Local Environment
LTo adhesion level insufficient
to prolong contact (< Ξ)
OR cell moves away for
other reason
Prolonged Surface Contact (Adhesion Effect)
Contact Cell is an LTo Cell
AND
Bind between cells is sufficient (> χ)
AND
LTo adhesion level sufficient
to prolong contact (> Ξ)
Contact Cell is an
LTo Cell
AND
Bind between cells is
sufficient (> χ)
AND
LTo adhesion level
insufficient to
prolong contact (< Ξ)
Distance between
cell and RET
Ligand Expressing Cell
< σ/2 + τ/2
Contact Cell is not
an LTo
OR
Bind between cell
& LTo cell is
insufficient (< χ)
Contact with Cell Expressing RET Ligand
1: Cell Speed falls somewhere between a lower bound specified by ω and an upper bound specified by ξ

(b) LTi

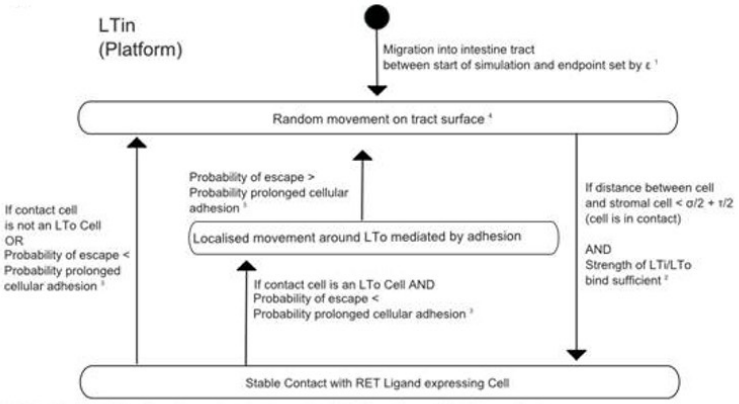


(c) LTo

Figure 6.2: Domain Model State Diagrams[1], [2]

	Parameter	Value
χ	Probability stable bind occurs on contact	50%
θ	Percentage of LTo cells expressing RET Ligand	*Unknown*
H	Percentage of RET Ligand Cells that are non-stromal	*Unknown*
ϖ	Hours Immature LTo remains active	72 hours
N	LTo Division Time	12 hours
ρ	Hours RET Ligands Expressed	72 hours
τ	LTin / LTi Cell Size	8 µm
σ	LTo Cell Size	24 µm
ω	LTin / LTi Speed Lower Bound	3.8 µm / min
ξ	LTin / LTi Speed Upper Bound	8.8 µm / min
ε	LTin Input Time	72 hours
γ	LTi Input Delay Time	0 hours
η	LTi Input Time	0 hours
φ	Chemokine Threshold	*Unknown*

Figure 6.3: Domain Model Biological Parameters[1], [2]



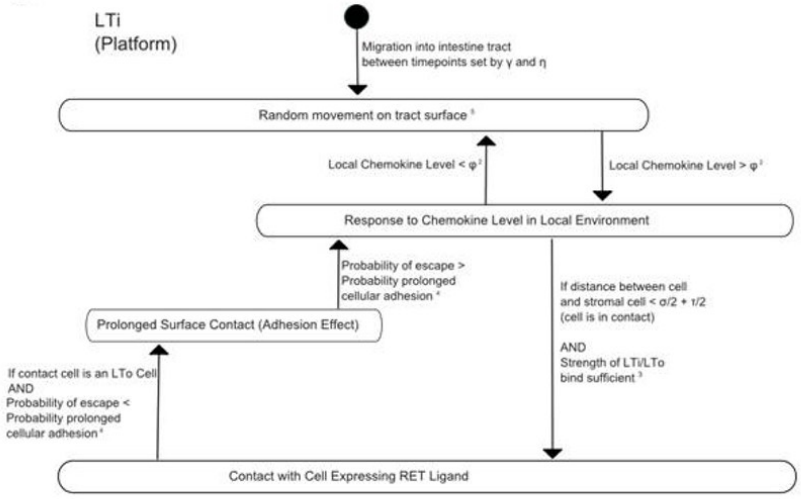
1: LTin cells migrate into the environment at a rate, ς , calculated through one of the below methods:
(a) If κ is 'linear', ς is (δ / ((24*60)*60) / Λ
(b) If κ is 'exponential', ς is λ raised to the power of the simulation time elapsed minus the current LTin cellularity
(c) If κ is 'square root', ς is λ multipled by the simulation time elapsed, raised to the power of 0.5, with the current LTin cellularity subtracted from the total.
For (b) and (c), to ensure the correct number of LTin cells is reached at E15.5 (δ), the constant λ has been calculated using δ

2: Strength of bind is a probability. If a probability is chosen which is greater than χ , stable bind is deemed to have occurred.

3: Probability of prolonged adhesion is calculated by (ν * Contacted LTo Adhesion Expression). Should the expression level be high, the probability is set to the threshold value stated in Ξ . This ensures some stochasticity remains, and cells may potentially ignore the adhesion effect.

4: In the domain model, cell speed falls somewhere between a lower bound specified by ω and an upper bound specified by ξ. These limits assume the cell is moving that distance per minute. In the model, no assumption is made that each simulation step will represent a minute of developmental time; it could be more or less than that. Therefore, the upper and lower bounds need to be calculated from the chosen number of seconds represented by each step (Λ):
The lower bound for the simulation run, Π , is calculated by ω / 60 * Λ
The upper bound for the simulation run, Θ , is calcualted by ξ / 60 * Λ

(a) LTin



1: LTi cells migrate into the environment at a rate, Ψ , calculated through one of the below methods:
(a) If Υ is 'linear', Ψ is (δ / ((24*60)*60) / Λ
(b) If Υ is 'exponential', Ψ is A raised to the power of the simulation time elapsed minus the current LTi cellularity
(c) If Υ is 'square root', Ψ is A multipled by the simulation time elapsed, raised to the power of 0.5, with the current LTi cellularity subtracted from the total.
For (b) and (c), to ensure the correct number of LTi cells is reached at E15.5 (ϕ), the constant A has been calculated using ϕ

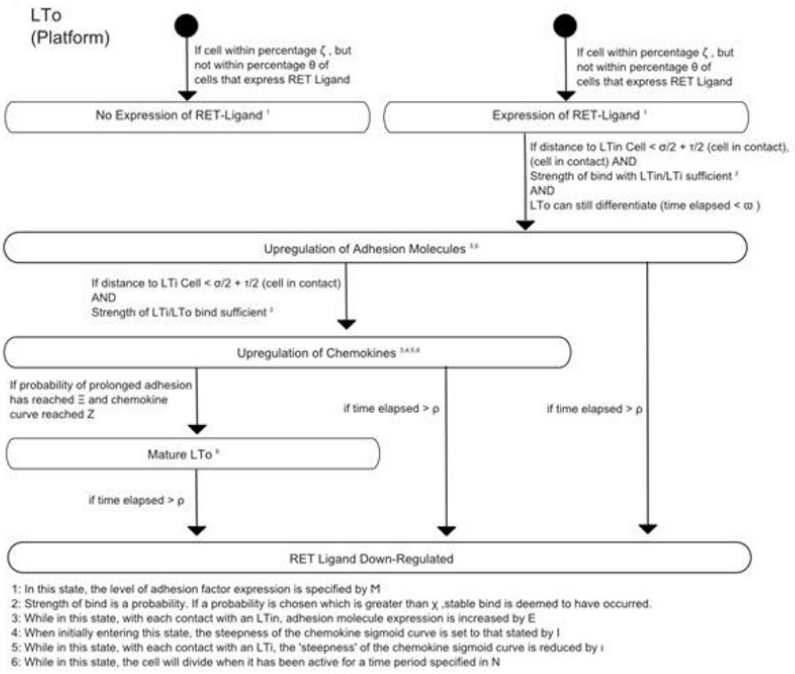
2: To ascertain chemokine level, the model will calculate the level in each 'gridsquare' around the cell. More detail in the environment description on how this is done. If the strongest level is greater than φ, the cell will move in that direction

3: Strength of bind is a probability. If a probability is chosen which is greater than χ ,stable bind is deemed to have occurred.

4: Probability of prolonged adhesion is calculated by (ν * Contacted LTo Adhesion Expression). Should the expression level be high, the probability is set to the threshold value stated in Ξ . This ensures some stochasticity remains, and cells may potentially ignore the adhesion effect.

5: In the domain model, cell speed falls somewhere between a lower bound specified by ω and an upper bound specified by ξ . These limits assume the cell is moving that distance per minute. In the model, no assumption is made that each simulation step will represent a minute of developmental time; it could be more or less than that. Therefore, the upper and lower bounds need to be calculated from the chosen number of seconds represented by each step (Λ):
The lower bound for the simulation run, Π , is calculated by ω / 60 * Λ
The upper bound for the simulation run, Θ , is calcualted by ξ / 60 * Λ

(b) LTi



(c) LTo

Figure 6.4: Platform Model State Diagrams[1], [2]

	Parameter	Simulation Variable	Value
ς	LTin Cell Input Rate	*Calculated*	79 cells / 100 steps
Ψ	LTi Input Rate	*Calculated*	97 cells / 100 steps
τ	LTin / LTi Cell Size	LTI_CELL_SIZE	6 px
σ	LTo Cell Size	LTO_CELL_SIZE	2 px
Π	Simulation Run Cell Speed Lower Bound	*N/A*	0
Θ	Simulation Run Cell Speed Upper Bound	MAX_CELL_SPEED	10
M	Initial Expression Adhesion Factors	INITIAL_ADHESION	0
Ξ	Linear Equation Slope	ADHESION_SLOPE	Calibrated to 1
E	Increase in Adhesion with each stable contact	ADHESION_INCREMENT	Calibrated to 0.05
ν	Adhesion Level Threshold	MAX_ADHESION_PROBABILITY	Calibrated to 0.65
φ	Chemokine Threshold	CHEMO_THRESHOLD	Calibrated to 0.3
B	Sigmoid Curve Adjustment	CHEMO_CURVE_ADJUST	3
I	Initial Curve Value	CHEMO_UPPER_ADJUST	Calibrated to 0.2
Z	Lower Curve Value	CHEMO_LOWER_ADJUST	Calibrated to 0.04
ι	Increase in expression with stable contact	INCREASE_CHEMO_EXPRESSION	Calibrated to 0.005
K	Circumference (Environment growth not implemented)	CIRCUMFERENCE	254
P	Length (Environment growth not implemented)	LENGTH	7303

Figure 6.5: Platform Model Simulation Parameters[1], [2]