August 2015

Performance Optimization and Statistical Analysis of Basic Immune Simulator (BIS) Using the FLAME GPU Environment

Shailesh Tamrakar
University of Wisconsin-Milwaukee

Performance optimization and statistical analysis of Basic Immune Simulator (BIS) using the FLAME GPU environment

by

Shailesh Tamrakar

A Thesis Submitted in

Partial Fulfillment of the

Requirements for the Degree of

Master of Science

in Engineering

at

The University of Wisconsin–Milwaukee

May 2015

Abstract

Performance optimization and statistical analysis of Basic Immune Simulator (BIS) using the FLAME GPU environment

by

Shailesh Tamrakar

The University of Wisconsin–Milwaukee, 2015
Under the Supervision of Professor Roshan M D'souza

Agent-based models (ABMs) are increasingly being used to study population dynamics in complex systems such as the human immune system. Previously, Folcik et al. developed a Basic Immune Simulator (BIS) and implemented it using the RePast ABM simulation framework. However, frameworks such as RePast are designed to execute serially on CPUs and therefore cannot efficiently handle large simulations. In this thesis, we developed a parallel implementation of immune simulator using FLAME GPU, a parallel ABM simulation framework designed to execute of Graphics Processing Units(GPUs). The parallel implementation was tested against the original RePast implementation for accuracy by running a simulation of immune response to a viral infection of generic tissue cells. Finally, a performance benchmark done against the original RePast implementation demonstrated a significant performance gain ($13\times$) for the parallel FLAME GPU implementation.

Table of Contents

List of Figures

List of Tables

List of abbreviations

ABM	Agent Based Model
BIS	Basic Immune Simulator
CTLs	Cyto-Toxic Lymphocytes
CUDA	Compute Unified Device Architecture
DCs	Dendritic Cells
FLAME	Flexible Large-scale Agent Modeling Environment
FSM	Finite State Machine
GPU	Graphics Processing Unit
IBM	Individual Based Model
LSD	Laboratory for Simulation Development
MΦ	Macrophages
MASON	Multi-Agent Simulator Of Neighborhoods
MATSim	Multi-Agent Transport Simulation
NKs	Natural Killers
ODE	Ordinary Differential Equation
PCs	Parenchymal Cells
PDE	Partial Differential Equation
SIMT	Single Instruction Multiple Threads
SMs	Streaming Multiprocessors
SPs	Streaming Processors
XMML	X-Machine Markup Language
XSLT	Extensible Stylesheet Language Transformations

Acknowledgements

I would like to express my deepest gratitude to my advisor Dr. Roshan M D'souza for providing me valuable guidance and support during my enrollment as a student at University of Wisconsin-Milwaukee. I truly appreciate his exemplary mentorship and insight that motivated me to work on this project.

I would also like to thank Dr. Paul Richmond from The University of Sheffield, UK for providing valuable technical support for FLAME GPU. Without his technical expertise, it would have been really difficult to complete this project.

Finally, I would like to thank my family for being by my side in every step, in good as well as bad times.

Chapter 1

Introduction

Humans live in an environment surrounded by pathogens such as viruses, bacteria, and fungi. Frequently, these pathogens enter the body and infect vital tissues and organs resulting in various diseases. The human body is equipped with a biological defense system called the immune system whose primary purpose is to fight invading pathogens and keep functional cells healthy. The Immune system can be sub-divided into two classes: Innate and Adaptive immune system. The innate immune system cells form the first line of defense which react to distressed signals released by cells infected by pathogens (1) and kills them upon contact. These are non-antigen-specific cells which respond to pathogens irrespective of their type (2). On the other hand, adaptive immune system cells which follow the innate immune system response are antigen-specific and proliferate exponentially to handle specific infections. Furthermore, unlike innate immune system cells, adaptive immune system cells being antigen specific, develop an immunological memory so that host response is much faster when same pathogen infects the functional cells of the body at a later date (3).

The immune system can modeled by two methods: equation based models (EBMs) and agent based models (ABMs). EBMs simply use set of ordinary differential equations (ODEs) and partial differential equations (PDEs) to track the temporal and/or spatial dynamics of quantities of interest such as average velocity of individual agents in a region, population size, and chemical concentrations.This technique works well in cases where the continuum assumption holds. It ignores the discrete nature of many dynamics system and is incapable of capturing interactions between autonomous individuals (agents) in such dynamic systems.

Agent based modeling, on the other hand, is a bottom-up approach (4; 5). It captures complex interactions between autonomous agents using a set of 'rules'. Agents themselves are interacting finite state machines with specific behaviors defined for each of the states. Rules also determine state transition. The collective behavior of these agents determines the macro-scale behavior of the system. The Basic Immune Simulator (6; 7) employs an ABM to study the interaction between innate and adaptive immunity. In this particular model, immune cells are modeled as agents which produce chemical signals based on what they detect from their proximal location and interact with other cells in a computer simulated environment.

The previous version of Basic Immune Simulator was implemented using an open source software library called Recursive Porous Agent Simulation Toolkit (RePast) (8; 9). The RePast toolkit is designed to execute serially on Central Processing Units (CPUs). Therefore, it is limited in its capability to handle large model sizes that are required to simulated realistic agent-based model simulations.

In this thesis we report on the parallel implementation of the Basic Immune Simulator (BIS) using the Flexible Large-scale Agent Modeling Environment on Graphics Processing Units (FLAME-GPU)(10). FLAME-GPU, as the name suggested, is a parallel computing framework designed to leverage the computing power of GPUs. To test the accuracy of our implementation, we ran several simulations of the BIS with same initial conditions on both the FLAME-GPU implementation as well as the original RePast implementation. Our results show that we can replicate results from the RePast implementation within statistical limits. Next, we conducted a performance benchmark of the FLAME-GPU implementation against the RePast implementation for varying model sizes. Our results show upto 13× speedup for models that have 20,000 agents.

In chapter 2, we provide a brief overview of the agent-based modeling technique. In chapter 3, we describe GPU-based parallel computing and the FLAME-GPU agent modeling framework. In chapter 4, we provide a brief overview of the human immune system. In chapter 5, we describe our implementation of the Basic Immune System agent-based model in the FLAME-GPU framework. In chapter 6, we provide the results of our accuracy and performance tests. Finally, in chapter 7. we provide conclusions and directions for future work.

Chapter 2

Complex systems and Agent Based Models

2.1 Complex systems

The world we live in is abundant with numerous Complex Adaptive Systems (CAS) that exhibit intricate behaviors. A complex dynamic system is composed of discrete autonomous individuals that interact with each other locally. These local interactions can be non-linear and stochastic. The collective/global behavior of the system *emerges* from the local interactions of individuals. An ant-colony is an example of a real world system that shows complex adaptive behavior. According to the study conducted by Deborah Gordon on ant colonies (11; 12), ants are classified based on specific task assigned to each group of ants in the colony: foraging, patrolling and nest maintenance. A group of ants who search and collect food to feed their offspring are termed foragers. They follow the trails of pheromones left by other forager ants to find their way to the food (11; 13). Pheromones, in this case, act as a medium for communication between ants. This communication between ants through pheromones leads to a formation of a highway of ants with two lanes (14). One lane comprise of group of ants bringing food to the nest, while the other lane has a queue of ants, sensing the trail of pheromones, to make their way to food source. Patrollers are the group of ants responsible for nest surveillance. They protect their nest from damage and invasion from other insects. Nest maintenance workers have the role of keeping trails free of obstructions such as sand particles that makes sensing of pheromones by foragers difficult. It has been found that the process of allocating task to group task is a continually changing process (11; 15; 16). The allocation of tasks depends upon the existing situation of an ant colony. For example, if a colony experiences a shortage of food, then there will be increased recruitment of foragers from other group of ants until the problem of shortage of food is resolved. However, the process of task allocation in an ant colony is not influenced by instructions passed from queen ant, but from local interactions between the ants itself through chemical or tactile communication (11). The overall behavior of the ant colony thus *emerges* from local interactions between ants, rather than from a central controller (11; 16).

There are other groups of animals such as flock of birds and schools of fish that show emergent behaviors (11; 17; 18) . For example, the direction of movement of an individual fish in a school of fish is influenced by direction of movement of its neighboring individuals (19; 20). The same is the case with the flight behavior of individual birds in a flock of birds (21; 11). Such coordinated behavior is not just limited to group of animals but also exhibited by phenomena that take place on daily basis such as traffic jams.

Traditional methods of modeling using ordinary differential equations (ODEs) and partial differential equations (PDEs) cannot be applied to such systems. The continuum assumption that forms the basis of ODEs and PDEs is not applicable because of the discrete and autonomous nature of individuals in the systems, and associated stochastic interactions. In the recent past Agent-based Modeling (ABMs) has been used successfully used to build computational models of CAS. The availability of commodity high performance computing processors such as graphics processing units (GPUs) has enabled the simulation of realistically sized models with the finest level of granularity.

2.2 Agent Based Models (ABMs)

Agent Based Models (ABMs) are computational models which are used to simulate repetitive interactions between autonomous agents to observe their overall effect to the system in which they reside. It is sometimes referred to as Individual Based Modeling (IBM) (22) because it primarily focuses on modeling individual agent variables that represent its characteristics rather than variables that describes the global state of the system. It adopts a bottom up approach where local, micro-scale interactions of agents gives rise to macro-scaled behavior of the system.

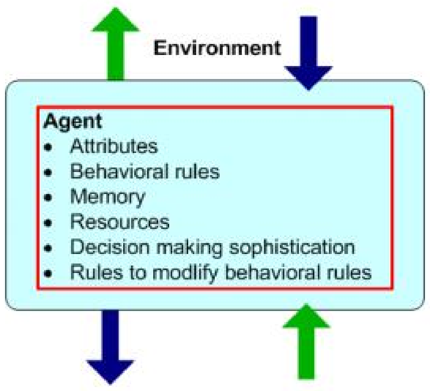

Environment
Agent
Attributes
Behavioral rules
Memory
Resources
Decision making sophistication
Rules to modlify behavioral rules

Figure 2.1: Definition of an agent

The definition for the term 'agent' (Figure 2.1) (23) in ABM is vague and there is no plausible compliance among the authors. Bonabeau stated that any independent component within a system can be considered called an agent (24; 23). Casti defined agents as individuals which must contain 'base-level rules' which governs the behavior of agents in response to its environment and 'rules to change the rules' to demonstrate adaptive behavior (25; 23). Jennings described agents as autonomous individuals having an ability to make independent decisions (26; 23).

Each agent is unique, autonomous and interacts with its local environment (27). The agents are called 'unique' because each agent within a system is different from each other in terms of characteristic variables such as its location, state and type

(27). Additionally, each agent is 'autonomous' because it is independent of other agents and executes its own sequence of rules (27). The behavior of agents is an outcome of its interactions with local environment which is limited to its neighboring agents and environment within certain radius of influence. This implies that agents interact locally and they are dependent upon state of neighboring agents rather than overall state of the system. One of the primary features of agents in Agent Based Models (ABMs) is the ability of agents to adjust its behavior based on its interaction with other agents or its response to what it senses from its environment. This kind of behavior is referred to adaptive behavior (27) which introduces randomness and stochasticity in ABMs.

2.3 History of ABMs

Agent Based Model (ABM) came into existence in 1940s but it was not extensively used until 1990s because of limitation of computational capability of early computers and unavailability of ABM modeling frameworks. The first Agent Based Model (ABM) called Cellular Automata (28) was developed by John Von Neumann in late 1940s. It consists of set of cells or grids in predefined states (29). The state of each cell is dependent upon the states of adjacent neighboring cells. Later in 1970s, John Conway, a British mathematician, used Cellular Automata to develop a model called Conway's Game of Life (30). Other notable examples of ABMs developed in between 1970s and 1980s include the study of housing segregation pattern development by Thomas Schelling (29) and Robert Axelrod's 'Prisoner's dilemma strategies' (31; 29). In late 1980s, Craig Reynolds developed the Boids model (32) to predict flocking behavior observed in birds. It was a first biological agent based model that incorporated social characteristics in agents. In between 1990s and 2000s, the modeling of complex system was simplified with introduction of frameworks such as Swarm, Netlogo and RePast. These frameworks were designed to simulate the Agent Based Models (ABMs) on a large scale. One notable example that simulated ABM on a large scale was Sugarscape model developed by Joshua M. Epstein and Robert Axtell which simulated the social phenomena such as transmission of disease, seasonal migration, pollution and sexual reproduction (33).

2.4 Applications of ABM

There are a wide range of applications of Agent Based Models (ABMs) in the field of social science, economics, archeology, biology and technology. Table 2.1 below shows the summary of applications of ABMs in different fields (23). ABMs that represent social scenarios define agents as people or group of people interacting with each other to maintain social relationships (23). ABMs in social sciences allow a modeler to model the people and their behavior to certain level of abstraction. The necessity of abstraction simplifies the model by including traits which produces desired behavioral outcome. The agents are assumed to be heterogeneous with each agent possessing distinct characteristics.

ABMs also have been widely used as a computational tool in economics to perform analysis. The agents, termed as economic agents, are homogeneous having same characteristics and behavioral rules (23). They possess a trait to pursue their objectives and adjust their behavior depending upon the economic situation. For example, people with limited economic resources would exhibit the trait of being thrifty.

Archeologists have been using ABMs to study the growth and decline of ancient civilizations due to social and environmental factors. The study on extinction of the Anasazi in the southwestern United States (34; 23) is an example of ABMs used in the field of archeology.

Industry	**Biology**	**Urban planning**	**Structure**	**Economics**	**Society and culture**
Supply chain management	Population dynamics	Pedestrian crowd model	Traffic congestion model	Trade networks	Ancient civilizations
Production operations	Cellular systems and processes	Emergency evacuation model	Electric power markets	Artificial financial markets	Regulative networks
Consumer market	Ecological networks				

Table 2.1: Summary of application of ABMs in different fields

Since ABMs capture micro-scale behavior in terms of rules, it is extensively used in the field of biology. This technique is particularly useful in situations where micro-scale behaviors can be experimentally studied and codified using rules. The system as a whole can be then be modeled as an ABM. Models of human immune system (23), tissue formation and morphogenesis (35; 36), spread of epidemics, population dynamics (37) and plant ecology (38) are few examples of ABMs in biology.

The use of ABMs has been in rise since 1990s to solve the problems in business and technology sector. The study of impact of publications in journals as opposed to conferences by researchers in the field of computer science (39) is one of the notable examples of contribution of ABMs in business and technology. Other examples include modeling of supply chain management and logistics, consumer behavior and organizational behavior (40).

2.5 Frameworks for ABMs

There are many frameworks to simulate Agent Based Models (ABMs) for wide range of applications. The details for available frameworks to simulate ABMs are described in a Table 2.2 (41) below.

Table 2.2: Available frameworks to simulate Agent Based Models (ABMs)

S.N.	**Framework**	**Description**	**Programming language**
1.	Ecolab (42)	It is designed, especially, for simulation of evolutionary dynamics. It possesses various tools to plot graphs and histograms.	C++
2.	FLAME GPU (10)	It is generic Agent Based Modeling system used to simulate complex system with large agent population in Graphics Processing Units (GPUs). It can model both discrete and continuous agents. It also has inbuilt 3D visualization tool to view simulations in real time.	C based script with optimized CUDA code
3.	LSD (Laboratory for Simulation Development) (43)	It is a programming language to develop simulation models for social sciences. It is also used in implementation of discrete-time simulation models.	C++ based LSD

S.N.	**Framework**	**Description**	**Programming language**
4.	MASON (44)	It is multi-agent simulation library to model discrete agents to perform simulations of generic ABMs. It has an optional visualization tool for both 2D and 3D models.	Java
5.	MATSim (Multi Agent Transport Simulation) [5]	It is a framework to simulate large-scale agent-based transport system.	Java
6.	NetLogo (45)	It is designed for simulating social and natural processes. It is an open-source library especially designed for people having limited programming experience.	NetLogo (extension of Logo programming language)
7.	Pandora: Agent Based Modelling Network for HPC (46)	It is a library developed to simulate large scale ABMs in High Performance Computing (HPC) environments. It is mostly used for social simulation research.	C++
8.	Swarm (41)	It is also used to simulate general ABMs. Its architecture allows a user to define agents different from each other.	Java, C

S.N.	Framework	Description	Programming language
9.	RePast (8)	It is an open-source library that can be employed for wide range of ABMs applications such as social sciences and biology. Its visualization tool is user interactive where user can optimize initial parameters for the simulation.	Java (RePastS, RePastJ), C++, C#, Python (RePastPy)
10.	StarLogo (47)	It is also a generic ABM framework to simulate real life processes such ant colonies, flocking of birds and traffic jams. It is an extension of Logo programming language where it allows a user to simulate large population of agents, termed as 'turtles', in parallel.	StarLogo (extension of Logo programming Language)

2.6 Agent Based Models (ABMs) vs Equation Based Models (EBMs)

ABM and EBM both share two common entities within a system: individuals and observables (5). Individuals are physical entities, such as atoms, present in a system that interact with each other by the influence of measurable variables called observables. The observables either represent characteristics of separate individuals such as velocity of particles inside the box or an entire system influenced by group of individuals (5). The pressure exerted by group of particles inside the box is an example of observables associated with a system.

The first difference between ABM and EBM is the way these models establish relationship between observables. Equation Based Models (EBMs) or Mathematical Models use a set of mathematical equations such as Partial Differential Equations (PDEs) and Ordinary Differential Equations (ODEs) to create a relationship between observables. Unlike EBMs which relates observables by mathematical equations, ABMs relate observables through specific behaviors imposed on separate individuals (5).

The granularity of observables also differentiates Agent Based Models (ABMs) from Equation Based Models (EBMs). There are two levels of granularity for observables: system level and individual level (5). System level observables, such as the pressure of gas, represent overall characteristics of a system. It is widely used in EBMs since it is easier to formulate such observables with a set of mathematical equations. Individual level observables, common in ABMs, focus on characteristics of each individual represented by agent behaviors.

2.7 Modeling cycle

The agents in Agents Based Models (ABMs) are designed to perform series of tasks during their life cycle. The tasks, they are assigned to execute must correlate with their behaviors in real life. It is not feasible to capture all the intricate details of real life behavior in the model. Therefore, it is necessary to simplify these behaviors to a certain level of abstraction so that the quality of outcome is preserved. The process of simplification must follow a modeling cycle to verify the results from the scientific model. The modeling cycle is comprised of five steps as shown in Figure 2.2 (27) and they are described below:

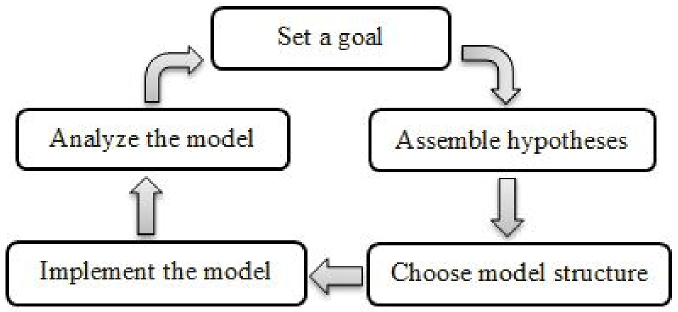


Figure 2.2: Modeling cycle for Agent Based Models

2.7.1 Set a clear goal

This step is the first step in the modeling cycle in which a modeler sets up certain goals that a model is designed to achieve. For example, a modeler is designing a system in which agents hunt for specific species of mushroom in a forest (27). The primary goal for this particular model is to identify edible mushrooms from poisonous ones. For this case, search for other plant species is not considered to be a primary goal of the model, thus, it can be filtered. Hence, setting up necessary goals even before the implementation of the model filters out the trivial details such as search for other plant species in a forest. However, this process has to be done on a recurring basis as it may require re-formulating the set goal due to intricate nature of complex systems. Thus, it requires maintaining a clear focus to achieve a desired goal (27).

2.7.2 Address hypotheses for processes and structures

The primary purpose of this step is to simplify the simulation model as much as possible. It is sometimes termed as brainstorming phase or simplification phase. In this step, many hypotheses are made for processes and structures that address the purpose or objective of the model. Hypotheses are formulated by a brainstorming technique by determining the factors that strongly influence the processes of the complex system with the help of influence diagrams and flow charts (27). It is important to make a simulation model simple because the preliminary understanding of the system alone cannot determine whether the factors contributing to specific processes and structures are important or not. The process of simplification is an iterative process which keeps on evolving as the factors affecting the system are added to model until a predefined goal is achieved. The simplification phase follows the cycle as illustrated in Figure 2.3 (27).

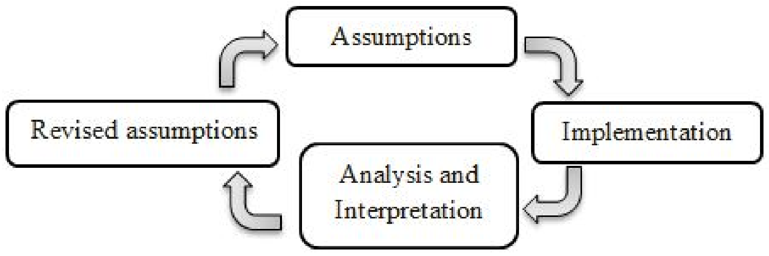


Figure 2.3: Simplification phase cycle

2.7.3 Choose entities, state variables and parameters

In this step, after the formulation of assumptions and hypotheses that simplify the model, the physical entities, state variables and parameters that constitute a model are determined (27). The physical entities represent the agents that reside in a simulated environment and execute specific rules of behavior. For example, mushroom hunters in a Mushroom Hunt Model are the physical entities that seek for specific species of mushroom in a forest. The state variables represent the characteristics of agents which directly address the behaviors of the agents based on its current state. In an example of Mushroom Hunt Model, if the count of mushrooms that hunter collected within some search radius reaches a maximum threshold value which is the maximum number of mushrooms each hunter can carry, then the hunter moves to a new location to search mushrooms (27). The count of mushroom gathered within a predefined search radius and the maximum threshold value are state variable and parameter that influences the behavior of agents.

2.7.4 Implement the model

In this step, theoretical description of model prepared in the previous steps is implemented by writing computer codes (Java, C++, and Python) or using available ABM frameworks such as RePast (8), FLAME GPU (10) and NetLogo (45). The primary objective of this step is to check if the initial model is useful by verifying the outcome of simplified assumptions and hypotheses.

2.7.5 Analyze, test and revise the model

This is the most demanding and time consuming step in a modeling cycle as a modeler has to run several tests to analyze the model. The analysis of model is done to test and improve the algorithms that represent the characteristics or behaviors of agents. The model is then revised to derive an optimal performance from the model. For example, the testing and analysis of different algorithms and parameters that increases the rate of finding mushrooms in a Mushroom Hunt model (27).

2.8 Design of ABMs

The informal descriptions such as ideas, assumptions and hypotheses, on processes and structures of a model need a formal and detailed description in order to formulate ABMs. Formulation of ABMs is the illustration of formal descriptions of the model in terms of algorithms, diagrams and equations that can be easily interpreted by others (27). This ensures that all important parts that constitute a model are included in a formal description of ABMs. Besides, it is also a basis for implementation of model in the form of computer programs that simulate them.

Equation Based Models (EBMs) use set of differential equations and parameters to formulate the model. Unlike EBMs, ABMs, which are complex and unpredictable, cannot be formulated by conventional differential equations. Apart of being complex and unpredictable, ABM formulations are vaguely descriptive with lengthy justifications and explanations. Thus, a standard procedure called ODD (Overview, Design concepts and Details) protocol is used to standardize the ABMs (27). This protocol allows a modeler to follow standard steps to include all the important characteristics of ABMs (agent types and its behavior) in a clear and concise way by organizing the information in consistent order. The first element of the protocol, 'Overview', provides the general description of model with respect to design and purpose of the model. The second element, 'Design concept', provides the necessary information on characteristics of ABMs. The last element, 'Details', depicts about the parts that make model formulation complete. The detailed description of each element of ODD protocol is explained below:

2.8.1 Purpose

The purpose of the model is to describe the problem statement that a specific model intends to solve. It makes decision process easier as it helps to select, prioritize, and validate the processes and structure of the model. Thus, a well-defined purpose or a problem statement gives a better understanding of a model.

2.8.2 Entities, state variables and scales

This element of ODD protocol creates a framework of the model by defining entities, state variables and scales. Entities refer to agents of different types which interact with each other within a fixed domain. The fixed domain is the environment where agents perform its actions. The environment can either be local or global: local environment affects a small group of agents at specific locations whereas global environment influences the characteristics of entire of agent population. An example of global environment is weather variables such as temperature and humidity that change with time (27). The environment represents a heterogeneous space and can be further classified into continuous and discrete space. The space is considered as continuous if each point in space has different environment variables. On the other hand, discrete spacing divides the environment into square grids or patches which prevents the inclusion of unnecessary variables thereby optimizes the computational performance of the model (27).

The variables that are governed by the properties and behaviors of agents are called state variables. Examples of state variable defined by the properties of agents are age, sex, type and memory. Similarly, searching behavior and learning algorithm of agents are examples of state variables defined with respect to behavior of agents. The state variable can be either static or dynamic in nature depending upon the type agent it represents. The variable that remains static i.e. it doesn't change with time during the course of simulation are called static variables. For example, location of a building within a city is an example of static state variable as it doesn't change with time. Dynamic variable, on the other hand, is constantly changing with time. The position coordinates of flock of birds is an example of dynamic variable. A variable such as distance between two agents which is calculated based on their location cannot be considered as a state variable. In other words, state variable is independent of state variables of other agents.

The scales for time and space of the model are defined within this protocol and they are termed as temporal and spatial scales respectively. Temporal scale is a representation of time in a model whereas spatial scale represents variation of environment in space. Temporal scale can be further classified into: temporal resolution and temporal extent. Temporal resolution represents discrete time steps in the form of a day, week or year. It helps to aggregate up all the events that occur before predefined time step. Temporal extent, on the flip side, represents the length of a simulation which is determined based on outcome expected from the model. Temporal extent is based on system level occurrences whereas temporal resolution is solely based on individual level phenomena (27). If a set of square grids or patches are used to represent the spatial scale of model, it is essential to define the size of each grids. The spatial effects between the grids are considered while spatial effect within each grid cell is ignored. For example, in urban dynamics each square grid represents a household. The activity inside each household is not taken under considered because it has no effect on urban patterns (27).

2.8.3 Process overview and scheduling

There are many processes that occur within a model and are collectively responsible for providing dynamics to the system. Therefore, processes occurring within a specific model refers to behavior or dynamics of agents that change their state variables. They are devised from an abstraction of real world processes to predict and analyze the complex systems. The processes that drive the model are explicitly defined by knowing the list and the order of behaviors that agents execute as simulation progresses and the changes it make to the environment (27). The order at which processes are executed is called scheduling. Like processes, order of execution of processes i.e. scheduling is also explicitly defined in the model. A clearly defined order provides a concise outline to the model, and hence, better qualitative results (27).

Apart from formulating processes, it is also necessary to track, observe and record the model's entities and their behaviors. This type of process is termed as 'observer process' which shows the model's status with the help of graphical displays and plots for tracking and recording the data such as agent count at specific time in simulation (27). This process helps a modeler to analyze the data extracted from the simulation to identify the shortcomings and, hence, optimize the model.

2.8.4 Design concepts

The design of Agent Based Models (ABMs) is based on an idea of 'conceptual framework' (27). It introduces basic concepts in standardized way that are essential for design of Agent Based Models (ABMs). Unlike the conceptual framework for Equation Based Modeling which characterizes the model's entities based on set of differential equations such as ODEs and PDES, the conceptual framework for ABMs characterizes the model by capturing important characteristics of ABMs. However, models designed from conceptual framework of ABMs can produce varying outcomes as opposed to EBM conceptual framework which always produces only one outcome by solving set of differential equations. The basic concepts that describe the characteristics of ABMs are briefly described below:

a. Emergence

This is one of the fundamental concepts that make Agent Based Models (ABMs) unique. It is obvious that, ABMs being complex systems, unpredictable and complex behaviors are expected when agents interact with each other or with their environment. Therefore, the concept of emergence helps to analyze the characteristics of agents that lead to emergent behavior. Besides, it also helps to distinguish emergent behavior from imposed behavior. For the model to be emergent, it has to fulfill following qualitative criteria: the emergent outcome must be independent and different from individual level properties (27).

b. Observation

Agent Based Model (ABM), being a dynamic system, produces different types of outcomes as a result of interaction between agents and their environment. The concept of observation is important when developing any kind of ABMs because it helps to analyze, interpret and validate the results obtained from the simulation, qualitatively as well as quantitatively (27). It allows a modeler to record, track and observe the state variables of each agents and verify if the agents are executing behavioral rules in correct order (27). Thus, it helps to make corrections to the behavioral rules of agents if they are not behaving as intended. Graphical displays and plots are the tools that are used in most of the ABMs to track, record, observe and analyze the results obtained from the simulation.

c. Sensing

The behavior of an agent is dependent upon the information it senses from other agents and its environment. The information, that an agent possesses, refers to the assumptions that modelers make to represent the characteristics of agents such as its state that determines if an agent is alive or dead. For example, in the Game of Life, this piece of information determines the current state of an agent. The agent with less than two live neighbors dies as a result of under population whereas an agent with two or more live neighbors continues to be alive (30). Therefore, it is essential to develop a technique to represent the way to sense information from other model's entities. First, it is necessary to determine the variables of other agents that contain information to be sensed by an agent of interest for its state change. Next, a mechanism, such as neighbor search algorithm, by which an agent of interest accesses information from its neighbors.

d. Adaptive behavior and objectives

Adaptive Behavior refers to the decisions that an agent make for its state transition in response to changes in the environment or within themselves to pursue some objectives. Agent decisions are governed by the set of rules or traits imposed on an agent which when executed lead to specific behavior. Therefore, it is essential to formulate the alternatives for decision making process and select one best alternative that fulfills the objective of the model. The alternatives (agent decisions) are examined by two basic concepts: adaptation and objectives (27). The adaptation concept helps in determining the decisions that agents execute to adapt to a constantly changing environment (27). The concept of objectives checks if the adaptive decisions made by agents meet their goal.

e. Prediction

The concept of prediction is essential to model adaptive behavior of agents as it allows a modeler to predict different outcomes as agents execute their adaptive traits in the model. Prediction models are sub-models, not a part of Agent Based Models itself, that simplifies the decision making process by testing different agent behaviors and finding the best one that produces a desired outcome that meets the objective of the model. There are two types of prediction for modeling agents' adaptive behavior: tacit and explicit prediction (27). Tacit predictions are implied and refer to the hidden assumptions in the model. Explicit prediction, on the other hand, predicts the future outcomes from previous experiences. Therefore, prediction models are essential in modeling adaptive behavior of agents and, hence, well designed ABMs.

f. Interaction

Interaction refers to the way agents communicate with each other or with its environment with the motive of exchanging information, sharing, using and competing for resources. The interaction between agents is affected by the state of an agent and takes place in either local or global scale (27). The local level interaction takes place between group of few agents at a specific location in the model whereas global interaction takes place at system level and affects all the agents present in the model. The interaction can be of two types: direct and mediated interaction. Direct interaction refers to the method of communication between agents as a result of physical contact between them. Mediated interaction, on the other hand, takes place when agents interact with each other through some mediator, such as pheromones trails that an ant senses in an ant colony.

g. Scheduling

ABMs have numerous processes that represent the behavior of agents. So, there is a necessity to maintain a right order at which these processes are executed to obtain outcomes that relates to real world systems. In a real world, these complex processes occurs simultaneously and continuously. The concept of scheduling models time and helps to simplify the model by representing these complex processes as discrete events that occur in a particular order, thereby, reducing the complexity of models and their outcomes (27).

h. Stochasticity

The concept of stochasticity represents the random processes, such as random motion of agents, in a simulation model. These random processes produce different outcomes every time the simulation model is executed because the random numbers generated in each simulation is different from the previous one. The use of random number generators, such as pseudo-random number generator (PRNG) (48), fulfills the need of randomness in any simulation models.

i. Collectives

Collectives are the group of specific agents that organize themselves in the model that affect individual agents as well the system (27). A tissue or an organ that perform a specific function in a body of an organism is an example of collective formed by group of specific cells. There are two ways to model collectives in ABMs. The first technique is to model behaviors of agents that allow them to organize with other agents to form collectives. The next approach is to consider collectives as an independent agent by defining its own state variables and behavioral rules explicitly. The state variables are defined based on state variables of individual agents that constitute a collective and behavioral rules for collectives are determined based on the way the individual agents are added and removed from a collective (27).

2.8.5 Initialization

It is often difficult to test viability of the model without assigning initial values to the state variables and global variables. State variables such as state of an agent which determines if it is alive or dead can be initialized by assigning value "1" if it is alive and value "0" if it is dead. In some ABMs, values of global variables affect the state of agents. In Mushroom Hunt Model, maximum number of mushrooms each agent can carry is an example of global variable. Additionally, the number of agents that take part in the simulation is also an example of initial condition.

2.8.6 Input data and sub-models

The input data is different from initialized values of variables and it is often used to input the values of environment variables such as temperature or market price that change with time. The input data are read from a input file into a simulation model.

Each process within a system can be deemed as a sub-model. The sub-models are independent of each other and can be designed and tested separately. They are listed in scheduling and are arranged based on the order of the processes (27).

Chapter 3

Flexible Large-scale Agent Modeling Environment (FLAME) GPU

The conventional Agent Based Model (ABM) frameworks such as RePast (8), Swarm (41) and NetLogo (45) are designed for single core CPU architectures. They are based on the concept of serial programming which means that agents execute their behavioral rules in serial manner i.e. one after another. While the computing power of CPUs has grown exponentially in the recent past, integrated circuit manufacturing techniques are scheduled to hit the physical limits of fabrication. Therefore, the growth in computing power is expected to plateau out. Consequently, simulating large scale ABMs (models with hundreds of millions of agents) with traditional ABM frameworks, running on serial computing platforms, is not expected to address issues of intractability.

An alternative way to address computational complexity is to use parallel computing architectures. However, parallel architectures require new algorithms. In the last decade, Graphics Processing Units (GPUs) have essentially democratized high performance computing. GPUs achieve high throughput by using several 'simple' cores with a simplified memory architecture. As long as algorithms adhere to the data-parallel computing model, GPUs deliver an order of magnitude advantage both in computing power and memory bandwidth as shown in Figure 3.1 (49)).

In a sense, parallel computing is a natural way to compute ABMs since the real life systems that they are used to represent are essentially parallel. Typically, the executions of each agents' processes are handled in parallel by different cores. Com-

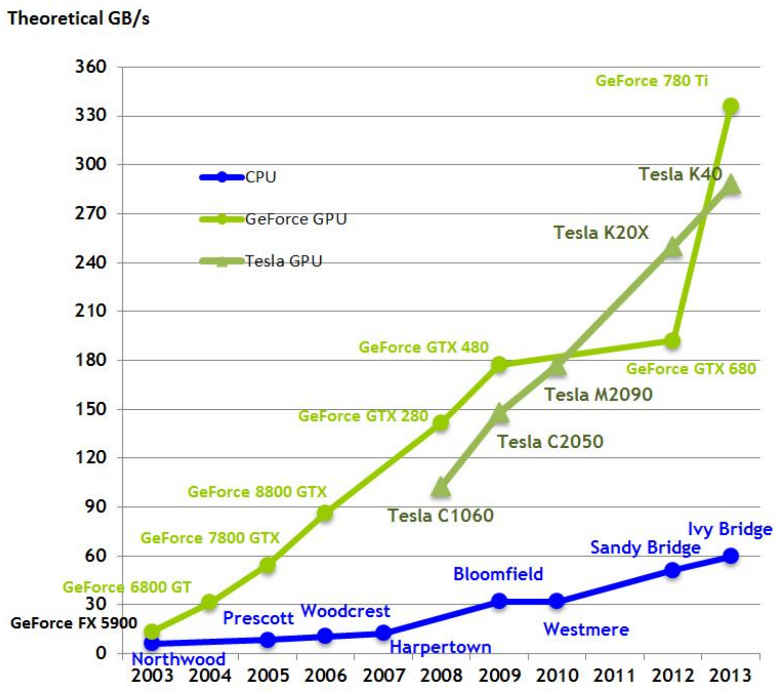


Figure 3.1: Memory bandwidth comparison CPU vs GPU

munication between agents is handled through shared memory that can be accessed by all cores. In this way, entire population of agents execute their behavioral processes at once which produces better computational performance than the processes that are executed serially.

An Agent Based Model framework called FLAME GPU, was developed at University of Sheffield by Dr. Paul Richmond to simulate large-scale ABMs in parallel for modeling a wide range of complex adaptive systems such as cellular systems and pedestrian crowds. FLAME GPU, is an extension to FLAME, a formal ABM specification schema, utilizes the parallel computing capability of Graphics Processing Units (GPUs). It maps formal agent specifications to C based optimized CUDA code. CUDA is the native programming language of GPUs by NVIDIA. It incorporates a feature to define multiple agent types, communication between agents via message boards and handles birth/death of agents. The advantages of using FLAME GPU framework are: First, it allows a modeler to implement agent behavioral rules and simulate ABMs without explicit understanding of parallel computing or the CUDA programming language. Second, it can simulate massive agent population with better computational performance as opposed to CPU alternatives. Third, visualization is easy to achieve since agent state variables are located in GPU memory and therefore, they can be rendered directly without any computational overhead.

3.1 Overview of GPU architecture

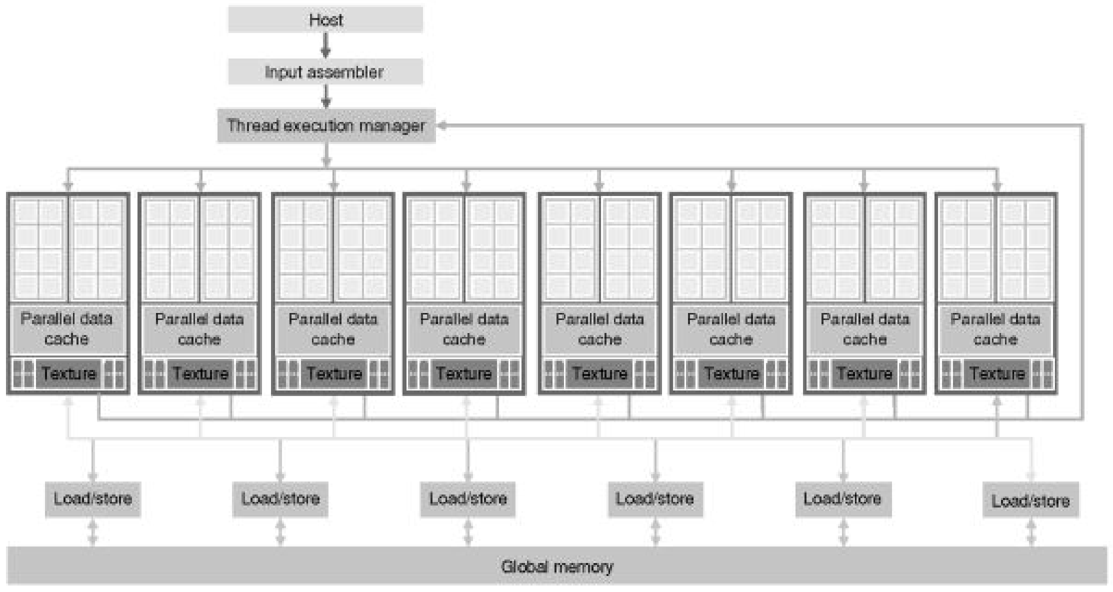


Figure 3.2: CUDA capable GPU architecture

Figure 3.2 (50) shows the hardware architecture of CUDA capable GPUs. At the lowest level, execution of a single thread is handled by a streaming processor (SP). Several processors are grouped into a Streaming Multiprocessor (SM). All SPs in a SM execute the same instruction in lockstep and share a instruction dispatch hardware. All threads executing on a SM can communicate directly through on chip user-controlled cache. Furthermore, threads across SMs can communicate through global memory which is equivalent to random access memory on CPUs. In addition, there are registers allocated to each thread that are used for storing local variables.

The memory control architecture is simplified compared to CPUs by restricting the ability to do recursive calls and code branching. On the flip side this allows a much higher level of memory bandwidth. For example, the G80 model of NVIDIA GPU has memory bandwidth of 86.4 GB/S.

3.2 Compute Unified Device Architecure (CUDA)

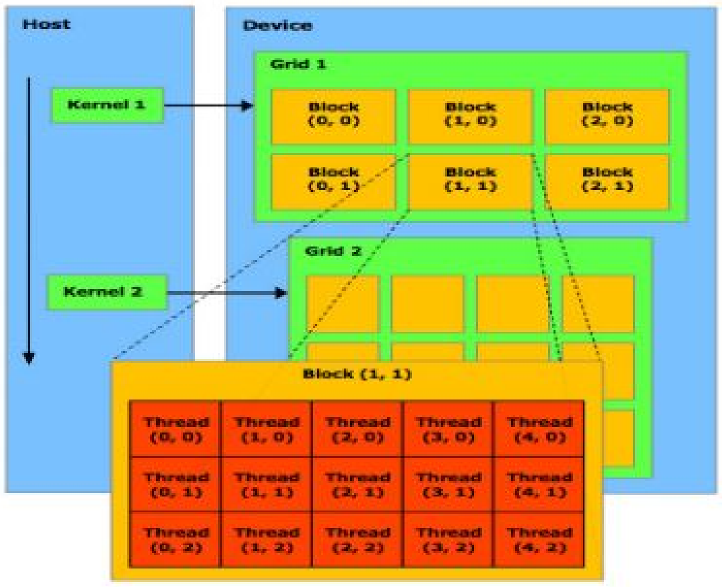


Figure 3.3: Basic units of CUDA

Programming in the GPUs made by NVIDIA is accomplished through a language specification called Compute Unified Device Architecture (CUDA). CUDA adapts the concept of SIMT: Single Instruction, Multiple Threads (50). The basic execution construct in CUDA is a kernel. Each thread executes the same kernel in parallel. At the software level, threads are organized into thread blocks. Each thread block (TB) is executed by a single SM with various threads in the thread block being handled by different SPs in the SM. Threads in a TB can be indexed either in 1-D, 2-D, or 3-D depending on the type of data being processed. Typically, the number of threads in a TB is much larger than the number of SMs. Threads in a TB are automatically scheduled for execution in blocks of thread warps. Similarly, the number of TBs is much larger than the number of SMs and the GPU automatically schedules the execution of TB on different SMs. At the highest level of execution, TBs are arranged in a grid layout. This layout can be 1-D,2-D, or 3-D to match the nature of the data being processed (51). Figure 3.3 illustrates this hierarchy.

3.3 CUDA memory types

There are five different type of memory in CUDA and they are briefly described below and illustrated in Figure 3.4 (51):

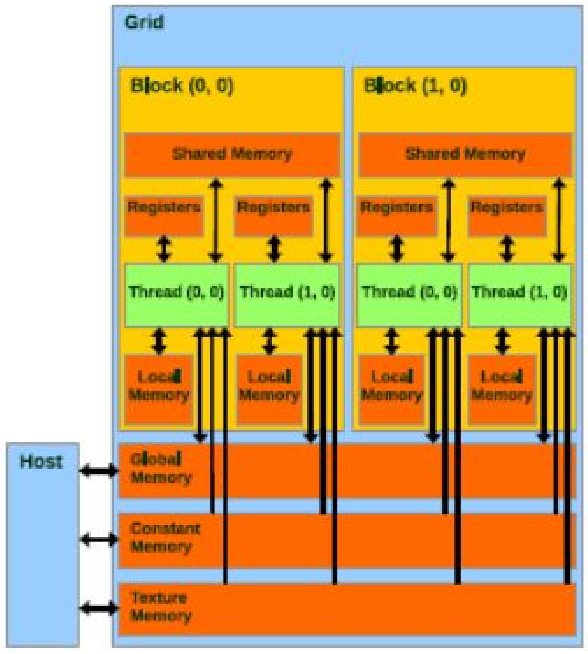


Figure 3.4: Memory types in CUDA

3.3.1 Global memory

It is a read and write memory which allows communication between threads located executing on SP located on different SMs. This memory is slowest on the GPU and typically requires coalesced read or write to achieve full bandwidth. In recent generation of GPUs, global memory is associated with L1 cache and therefore, performance of truely random access has increased substantially.

3.3.2 Shared memory

Shared memory is the user-controlled on chip-cache. All SPs in the SM have access to the same shared memory and therefore threads in a TB can communicate through share memory. However, threads in two TBs scheduled to execute on the same SM cannot communicate through shared memory becaused the share memory is not persistant between the scheduled execution of different TBs.

3.3.3 Registers

Registers are the fastest memory available in the GPU. The threads in Streaming Multiprocessors (SMs) are assigned a set of registers and they use registers to store data that is local to the thread such as counters that will need to be accessed frequently. The contents of registers cannot be shared directly by threads in TB.

3.3.4 Local Memory

Local memory stores the data overflow from registers. When a given thread has too many local variables, some of the variables are physcially stored in global memory through the local memory construct. Use of local memory can significantly affect performance.

3.3.5 Constant and Texture memory

Constant and texture memory are designed for read only operations. They are equipped with cache to speed up the data access for reading purpose. They are physcially located in global memory. Constant memory is used for storing variables that do not change in value over the execution life of a kernel. Texture memory data layout in a grid which can be 1-D, 2-D, or 3-D. Data in texture memory is parameterized and can be accessed even at locations inbetween grid points through suitable interpolation functions.

3.4 Process flow in CUDA

CUDA is a heterogeneous architecture which supports execution of both serial and parallel programs. The serial program generally executes the C code in the host (CPU) (51) whereas parallel programs called the kernel functions are executed in the device (GPU) (51). The execution of kernel function is done by threads present in the thread blocks. However, kernel function can only be invoked in a serial C code. The number of threads and thread blocks executing a kernel function is explicitly defined in serial C code when kernel function is called. The process flow in CUDA is illustrated by Figure 3.5 (52) and briefly described below:

Step 1: Transfer of data to device

Even before the data is copied from host (CPU) to device (GPU), it is essential to allocate memory for variables in both CPU and GPU. The memory is allocated in host using dynamic memory allocation function (*malloc*()) which is defined within C standard library. The memory allocation in GPU is done using CUDA API functions (49) (*cudaMalloc*()). After the allocation of memory is done in both CPU and GPU, the variables are transferred from host to device using CUDA API function, *cudaMemCpy*(), for parallel execution as shown by '1' in Figure 3.5. However, *cudaMemcpyHostToDevice* has to be specified within *cudaMemcpy* function which implies that data is being copied from host to device.

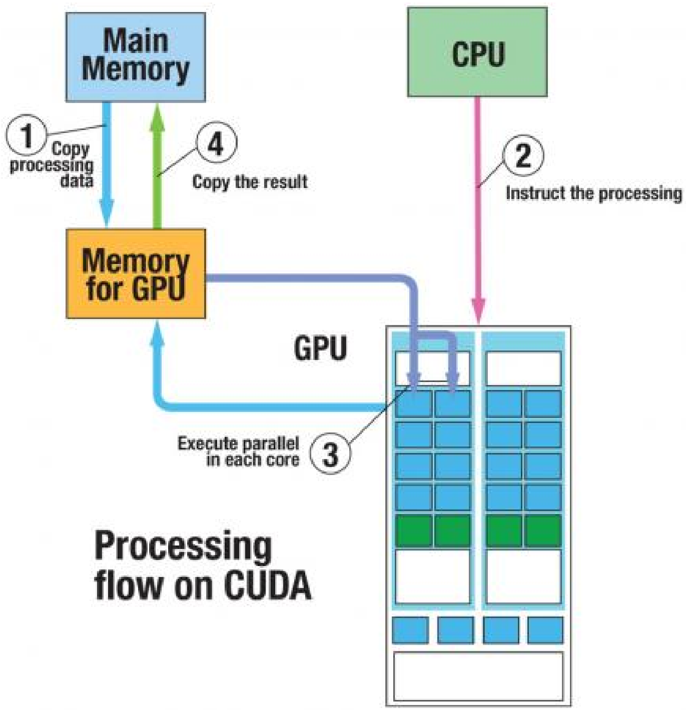


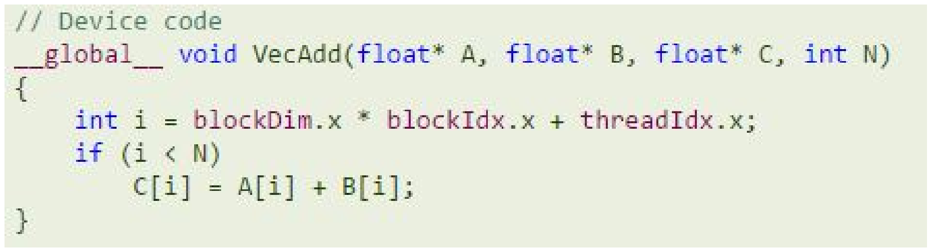
Figure 3.5: Process flow in CUDA

Step 2: Instruct the processing

In this step, a call to the kernel function is done from the host (CPU) for its parallel execution on the device (GPU). An execution configuration has to be specified in order to execute the kernel function. The execution configuration specifies the dimensions and layout of the TB grid, and the the dimensions and layout of the threads in the TB. Furthermore, function variables including pointers to data in global memory can be passed.

Step 3: Execute parallel processing

After the kernel function is invoked in step 2, it assigns a group of threads in a thread blocks with individual data variables to execute them in parallel. The kernel function has _global_ specifier before its name and has a *void* return type. The index for each thread is assigned within a kernel function by a inbuilt variable called "threadIdx" based on size of a thread block. An example of kernel function for a simple vector addition is shown in Figure 3.6 (49).



```
// Device code
__global__ void VecAdd(float* A, float* B, float* C, int N)
{
    int i = blockDim.x * blockIdx.x + threadIdx.x;
    if (i < N)
        C[i] = A[i] + B[i];
}
```

Figure 3.6: An example of kernel function

Step 4: Transfer of results to host

At the end of the execution of the kernel, data may be transfered back to the host for further analysis of results. Typically, this step (and also step 1) is executed only when absolutely necessary, for example, for archiving the results on a hardrive for check pointing. CUDA API function, *cudaMemcpy*(), copies the data from device (GPU) to host (CPU). However, *cudaMemcpyDeviceToHost* must be defined within *cudaMemcpy* function.

3.5 High level overview of FLAME GPU

FLAME GPU is a template based framework which maps the formal definition of agents to the simulation code (10). It is governed by X-machine Mark-up Language (XMML) with a set of XML schemas to verify the syntax of XMML model files. A typical XMML model file consists of definition of agents including transition functions, message types and layers which handle the order in which the simulation model is executed. Agents are defined based on the concept of X-Machine (53; 54), which is an extension of Finite State Machines (FSM) with internal memory. X-Machine agents are capable of communicating with other agents by iterating through message list from a message board. Transition functions, specified within agent definition, handles the state change of agents based on state transition rules. After the transition functions causes the state change of agents, they update their internal memory which can be either used as an input to iterate through the message list, or as an output which is passed as a message for other agents to read.

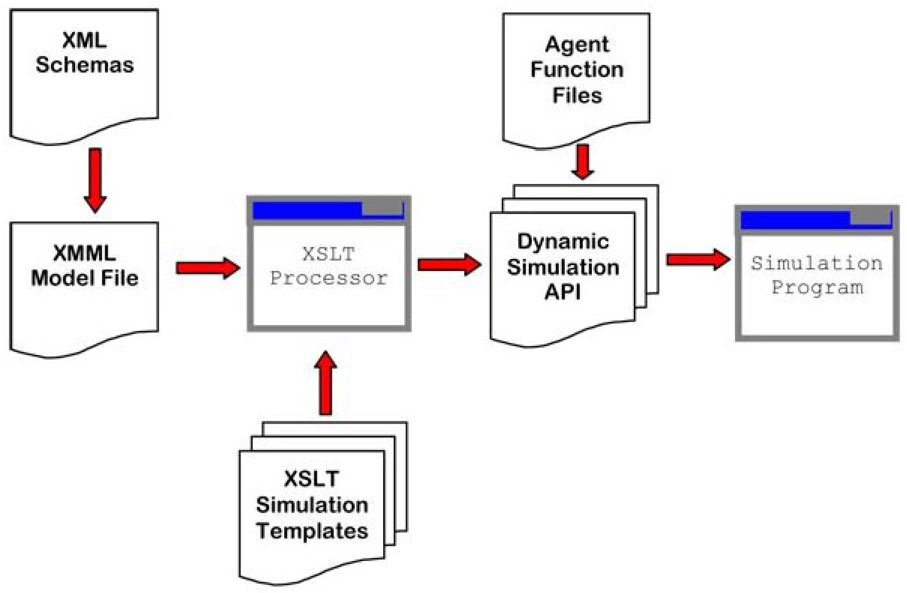


Figure 3.7: FLAME GPU process flow

Figure 3.7 (55) shows the process flow for the generation of compilable FLAME GPU simulation code. As shown in Figure 3.7, XMML model file along with Extensible Stylesheet Language Transformation (XSLT) templates is processed through XSLT processor to generate a compilable simulation code along with agent data structures, agent and message API functions (56). Two XMML schemas are used to validate the syntax of XMML model file: XMML Base Schema verifies the syntax of base XMML model file whereas GPU XMML Schema includes any additional GPU specific model parameters (56). The simulation model can be run in either visualization or console mode. The visualization mode allows a user to view the simulation in real time while the console mode allows a user to generate XML output files for predefined number of iterations.

3.6 Agent model specification in FLAME GPU

A skeletal layout of agent definition in XMML model file is shown in Figure 3.8 (55). X-machine agents are defined within single element of `xagents`. A `gpu:xagent` element contains name of an agent, its optional description, internal memory variables of agents, set of agent transition functions, set of states, type and the buffer size as shown in Figure 3.8.

```
<xagents>
    <gpu:xagent>
        <name>AgentName</name>
        <description>optional description of the agent</
            description>
        <memory>...</memory>
        <functions>...</functions>
        <states>...</states>
        <gpu:type>continuous</gpu:type>
        <gpu:bufferSize>1024</gpu:bufferSize>
    </gpu:xagent>
    <gpu:xagent>...</gpu:xagent>
</xagents>
```

Figure 3.8: Skeletal layout of agent definition in XMML model file.

The type refers to the way the agents are represented in spatial domain of the simulation model and are of two types: continuous and discrete. Discrete agents are represented as grids and can only move from one grid to the another as in Cellular Automata. Continuous agents, on the other hand, are allowed to move freely in continuous space. As the memory has be predefined in the GPUs, `bufferSize` determines the maximum number of specific agent types that take part in the simulation.

3.6.1 Agent memory

An agent memory consists of state variables of each agent which helps to store the information carried by it such as its position coordinates in x, y and z direction. Each agent variable consist of type (`int, float or double`) along with a unique `name` of agents. The available versions of FLAME GPU is capable of storing single memory variables only and the default values of these variables are always '0' unless any specific values are assigned to it in initial XML input file. Figure 3.9 (55) shows an example of agent memory defined for its position coordinates.

```xml
<memory>
    <gpu:variable>
        <type>int</type>
        <name>id</name>
    </gpu:variable>
    <gpu:variable>
        <type>float</type>
        <name>x</name>
    </gpu:variable>
    <gpu:variable>
        <type>float</type>
        <name>y</name>
    </gpu:variable>
    <gpu:variable>
        <type>float</type>
        <name>z</name>
    </gpu:variable>
</memory>
```

Figure 3.9: An example of definition for agent memory variables

3.6.2 Agent functions

Agent functions contain the definition of all the transition functions which handle the state change of agents. They are defined within `functions` element and it must include at least one agent transition function. An example of agent function definition is shown in Figure 3.10 (55).

As shown in the Figure 3.10, it consists of non optional and unique function name, an optional description of transition function, current state of agents, next state that agents attain, an optional single message inputs, an optional single message outputs, an optional agent outputs, an optional global conditions, an optional condition, a

```
<functions>
    <gpu:function>
        <name>func_name</name>
        <description>function description</description>
        <currentState>alive</currentState>
        <nextState>dead</nextState>
        <inputs>...</inputs>
        <outputs>...</outputs>
        <xagentOutputs></xagentOutputs>
        <gpu:globalCondition>...</gpu:globalCondition>
        <condition>...</condition>
        <gpu:reallocate>true</gpu:reallocate>
        <gpu:RNG>true</gpu:RNG>
    </gpu:function>
</functions>
```

Figure 3.10: An example of agent function definition

boolean (true or false) for reallocation and random number generation (55).The current state of agents defined within `currentState` acts as a filter and applies agent transition function to agents possessing the current state. The next state defined in `nextState` element is the state that an agent will after its state transition. The list of states that an agent attains is defined in `states` element. The `gpu:reallocate` element decides whether to keep or remove the agents from the simulation in the case of death of agents. The `gpu:reallocate` element can be either true or false: if it is defined as `true`, then the dead agents are removed from the simulation whereas if defined `false`, agents continue to be in a simulation model even if they are dead. The element `gpu:RNG` when initialized as `true` passes an array to agent transition function to generate random numbers in a simulation model.

1. Agent function message inputs and outputs

The element `inputs` in an agent function indicates that an agent function will iterate through message list located in global memory of the GPUs. It consists of non optional message name defined within `messageName` element and message name must be same as the name defined within `messages` element in XMML model file.

```
<inputs>
    <gpu:input>
        <messageName>message_name</messageName>
    </gpu:input>
</inputs>

<outputs>
    <gpu:output>
        <messageName>message_name</messageName>
        <gpu:type>single_message</gpu:type>
    </gpu:output>
</outputs>
```

Figure 3.11: An example of agent function inputs and outputs

The element `outputs` in an agent function is responsible for writing of messages from agents to a message board for reading by other agents. It consists of element `messageName` and `type`. The name in `messageName` must be exact to name of message defined in `messages` element. The `type` element can be either `single_message` or `optional_message`: `single_message` indicates that the agents executing agent transition function output only one message at a time whereas `optional_message` either outputs a single message or no message. If agent fails to output a message, the value of message variable is set to '0' by default. An example of agent function message inputs and outputs is show in Figure 3.11 (55).

2. Agent function X-agent outputs

The element `xagentOutputs` handles the birth of an agent. It consists of sub-element `gpu:xagentOutput` within which `xagentName` and `state` are defined. The name of an agent which is to be reproduced or added is specified within `xagentName` and its state is defined in `state` element which corresponds to the state of newly added agents. An example of agent function for xagent outputs in shown Figure 3.12.

```
<xagentOutputs>
    <gpu:xagentOutput>
        <xagentName>agent_name</xagentName>
        <state>alive</state>
    </gpu:xagentOutput>
</xagentOutputs>
```

Figure 3.12: An example of agent function inputs and outputs

3. Global function conditions

```
<gpu:globalCondition>
    <lhs>
        <agentVariable>movement</agentVariable>
    </lhs>
    <operator>&lt;</operator>
    <rhs>
        <value>0.25</value>
    </rhs>
    <gpu:maxItterations>200</gpu:maxItterations>
    <gpu:mustEvaluateTo>true</gpu:mustEvaluateTo>
</gpu:globalCondition>
```

Figure 3.13: An example of global function condition

A global function condition, defined within `gpu:globalCondition` element, acts as a switch to identify whether the agent transition function is applied to all or none of the agents. It consists of logical statements which are formulated using sub-elements: `lhs`, `operator` and `rhs`. The agent variable that need to tested are kept in `lhs` element. The `rhs` contains reference values or variables against which agent variables are tested. The `operator` element acts as a medium to compare `lhs` and `rhs` element. It also consists of `maxItterations` and `mustEvaluateTo` element. The element `maxItterations` allows to define maximum number of iterations to execute agent functions without being affected by global function condition. The `mustEvaluateTo` element drives the global function condition and its value is set to either `true` or `false`. If the value is set to `true`, global function condition is applied to all the agents whereas `false` value will not apply global function condition to any of the agents. An example of global function condition is shown in Figure 3.13 (55).

4. Function conditions

```
<condition>
    <lhs>
        <agentVariable>variable_name</agentVariable>
    </lhs>
    <operator>&lt;</operator>
    <rhs>
        <condition>
            <lhs>
                <agentVariable>variable_name2</agentVariable>
            </lhs>
            <operator>+</operator>
            <rhs>
                <value>1</value>
           </rhs>
        </condition>
    </rhs>
</condition>
```

Figure 3.14: An example of function condition with recursive `condition` element

The function condition is similar to global function condition but only difference between them is function condition is applied to agents that satisfy conditions applied to it. Like global function condition, it also consists of logical statements which are formulated by three sub-elements: left hand side state (`lhs`), right hand side statement (`rhs`) and an `operator`. The agent variables, constant values and recursive conditions are defined by `agentVariable`, `value` and `condition` element inside `lhs` or `rhs` element. Agent variables must be derived from the list of variables defined in `memory` element. An example of function condition with recursive `condition` element defined inside `rhs` element is shown in Figure 3.14 (55).

3.6.3 Agent states

Agent state lists all the state transitions that an agent makes during the course of the simulation. It is defined within `states` element in XMML model file with a unique and non optional `name`. Besides, initial state of an agent also needs to be defined which corresponds to its state during the start of the simulation. It is defined within `initialState` of `states` element. Figure 3.15 (55) is an example of listing of agent states (dead or alive).

```
<states>
    <gpu:state>
        <name>alive</name>
    </gpu:state>
    <gpu:state>
        <name>dead</name>
    </gpu:state>
    <initialState>alive</initialState>
</states>
```

Figure 3.15: An example of listing of agent states

3.6.4 Agent messages

Messages act as a bridge of communication between agents which allows agents to interact with each other. The agent variables which influence agents for its state change are passed as messages variables and are stored in memory board located in global memory of GPU which are accessible by all the agents. A skeletal framework for defining agent messages is shown in Figure 3.16 (55).

As shown in the Figure 3.16, it consists of unique and non optional `name` along with optional `description` of the message, a list of message variables (`variables`), partitioning type (`partitioningType`) and buffer size (`bufferSize`). The partitioning of message variables are of three types: non partitioned (`partitioningNone`), discretely partitioned (`partitioningDiscrete`) and spatially partitioned (`partitioningSpatial`)

```
<messages>
    <gpu:message>
        <name>message_name</name>
        <description>optional_description</description>
        <variables>...</variables>
        ...<partitioningType/>...    //replace with a
           partitioning type
        <gpu:bufferSize>1024</gpu:bufferSize>
    </gpu:message>
    <gpu:message>...</gpu:message>
</messages>
```

Figure 3.16: A skeletal framework for agent messages

(55). These partitioning techniques ensure that message variables are made available to agents accessing message variables in an optimal way. The `bufferSize` element determines the maximum number of messages that exist in a simulation model.

1. Message variables

Message variables contain vital information to allow communication between agents. The agent variables that influence the state change of other agents are defined as message variables. They are defined in `variables` inside `messages` element and are specified in a similar way as agent variables as shown in Figure 3.9.The only difference between agent and message variable is the need of certain variable name and type for specific message partitioning technique. For instance, discrete partitioning type requires position coordinates in x and y direction of type `int`. While, spatial partitioning needs three variables of type `float` in x, y and z direction. Non partitioned technique, on the other hand, has no limitation on type of message variables.

2. Non partitioned messages

Non partitioned messages require brute force search algorithm (57) which allows an agent to iterate through all the message lists. It has an order of $O(n^2)$ and it is computationally expensive when the message list is very large. However, this technique is efficient compared to spatial partitioned technique for small number of messages since it requires less overhead or setup (55) . It is defined as `<gpu:partitioningNone\>` inside `gpu:message` element in XMML model file.

3. Discrete partitioned messages

It is specifically used for non mobile discrete agents (Cellular Automata) as a medium of communication between other discrete agents (55). An example to define discrete partitioned messaging is shown in Figure 3.17 (55).

```
<gpu:partitioningDiscrete>
    <gpu:radius>1</gpu:radius>
</gpu:partitioningDiscrete>
```

Figure 3.17: An example of definition of discrete message partitioning

The `gpu:radius` element consists of constant value for the range of message iteration. When value of `radius` is defined as '0', the message iteration function returns a single message. However, when `radius` value is greater than '0', message iteration function seeks for messages in discrete neighboring grids in both x and y direction which are at a distance equal to `radius`. For example, if the value of `radius` is '1', the message iteration function will iterate through 9 messages (55) in 2-D space. Furthermore, message iteration function for discrete partitioning demands two message variables (`x, y`) of type `int` as input arguments.

4. Spatially partitioned messages

The spatially partitioned messaging technique is especially designed to gather messages from continuously spaced agents in a simulation model. This method is based on the concept of particle systems (58) which divides the simulation into uniformly sized grids. It is particularly efficient compared to non partitioned messages when number of messages is very large because this method iterates through the messages located in nearest neighboring grids rather than iterating through entire message list. It bins the agents messages belonging to same grid based on the grid index and sorts the messages by fast radix sort algorithm. A scatter kernel is used to find first and last index of agent messages (Figure 3.18) in sorted list to iterate through 9 messages (2D) or 27 messages (3D). An agent message within predefined radius of influence is considered for communication.

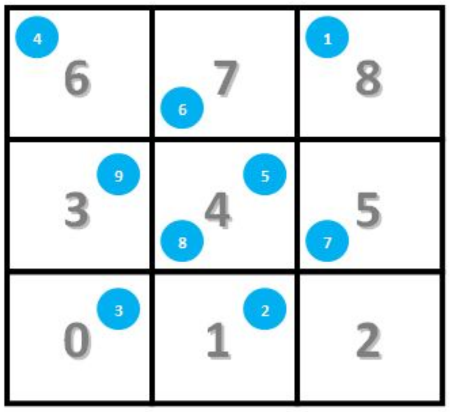


Figure 3.18: A schematic representation of 2D spatial partition technique

It consists of two sub-elements: a radius which determines the range of message iterations and a set of minimum and maximum bounds which represents the size of simulation domain within which agent messages exist. If agent messages lie outside the environment bounds, then they are bounded to nearest possible location in simulation domain (55). The message iteration function for this method required three agent variables (x, y, z) of type `float`. An example for defining 3D spatial partitioned messaging technique is shown in Figure 3.19 (55).

3.6.5 Function layers

Function layer handles the order in which the agent functions are executed in the simulation. The agent functions defined within function layers are executed for each iteration when the simulation is run for predefined number of iterations. An example

```
<gpu:partitioningSpatial>
    <gpu:radius>1</gpu:radius>
    <gpu:xmin>0</gpu:xmin>
    <gpu:xmax>10</gpu:xmax>
    <gpu:ymin>0</gpu:ymin>
    <gpu:ymax>10</gpu:ymax>
    <gpu:zmin>0</gpu:zmin>
    <gpu:zmax>10</gpu:zmax>
</gpu:partitioningSpatial>
```

Figure 3.19: An example of definition of spatial message partitioning

for definition of function layers is shown in Figure 3.20(55).

```
<layers>
    <layer>
        <gpu:layerFunction>
            <name>function1</name>
        </gpu:layerFunction>
        <gpu:layerFunction>
            <name>function2</name>
        </gpu:layerFunction>
    </layer>
    <layer>
        <gpu:layerFunction>
            <name>function3</name>
        </gpu:layerFunction>
    </layer>
</layers>
```

Figure 3.20: An example of definition of spatial message partitioning

As shown in the figure 3.20, it consists of `layers` element within which multiple `layer` sub-element may be defined. The `layer` can include multiple `gpu:layerFunction` element within which agent function name is defined. The agent functions which are independent of each other must to defined within a same `layer` whereas agent functions which share messages with each other are defined in a different `layer`. In the example shown above, `function1` and `function2` are defined within a same layer which shows that they are independent of each other.

3.6.6 Initial XML agent data

It consists of numerical values of agent variables that are passed as input before that start of the simulation. Agents' information are initialized within `states` element along with an iteration number (`itno`) set to '0' which is updated at every time step.

```
<states>
    <itno>0</itno>
    <xagent>
        <name>AgentName</name>
        <id>1</id>
        <x>21.088</x>
        <y>12.834</y>
        <z>5.367</z>
    </xagent>
    <xagent>...</xagent>
</states>
```

Figure 3.21: Initialization of agent variables for a single agent

The `xagent` element inside `states` element contains name and initial values for agent variables. The name of an agent and type of value assigned to it must be exactly same as defined in XMML model file. Figure 3.21 (55) shows an example of values assigned to each agent variables for a single agent.

3.6.7 Agent function scripting in FLAME GPU

Agent function script in FLAME GPU uses C based syntax to translate theoretical behavioral rules of agents into a computer code. In FLAME GPU, the change in behaviors of agents are governed by change in internal memory of agents when it iterates through messages of neighboring agents in a message iteration function. The creation of new messages as well as new agents also influence behavior of agents. The agent behavioral rule is defined for an individual agent in a specific state and it is assigned to all the agents in same state to run in parallel.

The data structures associated with agents and messages are dynamically created

```
__FLAME_GPU_FUNC__ int position_update
(xmachine_memory_myAgent* xmemory,
 xmachine_message_location_list* location_messages)
{
    int count;
    float avg_x, avg_y, avg_z,

    /* Get the first location messages */

    xmachine_message_location* message;
    message = get_first_location_message(location_messages);

    /* Loop through the messages */
    while(message)
    {
        if((message->id != xmemory->id))
        {
            avg_x += message->x;
            avg_y += message->y;
            avg_z += message->z;

            count++;
        }

        /* Move onto next location message */

        message = get_next_location_message(message,
           location_messages);
    }

    if (count)
    {
        avg_x /= count;
        avg_y /= count;
        avg_z /= count;
    }

    xmemory->x += avg_x;
    xmemory->y += avg_y;
    xmemory->z += avg_z;

    return 0;
}
```

Figure 3.22: Agent function script using non partitioned messaging to find average position of agents

and are defined in header file `"header.h"`. The agents variables are defined in a structure named `xmachine_memory_agent_name` where `agent_name` refers to name of an agent defined in XMML model definition (55). Likewise, message variables are included within `xmachine_message_message_name` structure. The header file `"header.h"` also contains structure of arrays that store agent and message list. Agent lists are created under the name of `xmachine_memory_agent_name_list` and message list are created with `xmachine_message_message_name_list` structure (55). These data structures are passed as arguments to agent functions and simulation API functions. The simulation API function could be message iteration function, message or agent addition function.

All the agent function declarations have a prefix `__FLAME_GPU_FUNC__` followed by agent function name and a set of function arguments defined within a parenthesis. The function name must be exactly same as defined in XMML model file. Additionally, the function arguments may consist of pointers which provide an access to agents, agents list or messaging list data structures and depends upon agent function definition in XMML model file. The function argument may also contain `RNG` pointer which generate random numbers in the simulation if `gpu:RNG` element is defined as `true` in XMML model file.

The body of an agent function consists of C based script that describes the rules of behavior for agents. Since the behavior of an agent is influenced by its neighboring agents, it also has a simulation API function for message iteration which allows an agent to access message variables of neighbors. The process for accessing message variables has three basic steps: First, a pointer is defined within agent function which provides an access to data structure containing message variables. Second, array containing first group of messages are loaded into shared memory by `get_first_message_name_message (arguments,...)` API function. Finally, a simulation API function `get_next_message_name_message (arguments,...)`, which reads messages in shared memory sequentially, is called inside a `while` loop until first group of messages present in shared memory is exhausted. After the first group of message list is exhausted, `get_next_message_name_message` function loads next group of messages into shared memory.

An example of agent function script using non partitioned messaging to find the average position of the agents is shown in Figure 3.22 (55). As shown in the figure above, the agent function name is `position_update` with two function arguments: a `xmemory` pointer which has an access agent data variables within `xmachine_memory_myAgent` structure and `location_messages` pointer which accesses message list present in `xmachine_message_location_list` structure. In the example illustrated above, the body of an agent function contains a C based script to calculate the average location of an agent based on position coordinates of its neighboring agents. It has a pointer `messages` which accesses to message variables i.e. position coordinates of neighboring agents. This pointer is then assigned to simulation API function, `get_first_location_message(location_messages)`, which takes a pointer to a message list as a function argument. This API function loads the first group of location messages into shared memory and these messages are read sequentially inside a `while` loop by `get_next_location_messages(message, location_messages)` assigned to `message` pointer, with two function arguments. Three variables `avg_x`, `avg_y` and `avg_z` of type `float` is defined in the beginning of agent function script to store the average value of position coordinates in x, y and z direction. The average values for position coordinates are calculated by summing the message variables having position coordinates inside a `while` loop which is then divided by total count of neighboring agents. Finally, calculated average values are assigned to agent variables which updates the internal memory of agents.

3.6.8 FLAME GPU simulation templates

FLAME GPU consists of number of simulation templates that generate dynamic simulation code. These templates are defined based on Extensible Stylesheet Transformations (XSLT) (55) and are of following types:

1. `header.xslt`

This template generates a header file `"header.h"` which contains agent and message data structures as well as agent and simulation function prototypes.

2. `main.xslt`

It generates a source file named `"main.cu"` which contains an entry point for execution of simulation. It also handles command line arguments, such as input XML file name and number of iterations.

3. `io.xslt`

It generates a source file, `"io.cu"`, which handles the reading of agent data variables from input XML file as well as write updated agent data variables into an output XML file.

4. `simulation.xslt`

It creates a source file `"simulation.cu"` for host (CPU) that is responsible for transferring agent data to and from device (GPU). It also includes calls to CUDA kernel functions that execute agent functions.

5. `FLAMEGPU_kernels.xslt`

It creates a CUDA header file `"FLAMEGPU_kernels.cu"` that contains CUDA kernel functions such as partitioning function which allocates each agent to its respective grid.

6. `visualization.xslt`

It creates an OPEN GL source file `"visualization.cu"` accompanied by header file `"visualization.h"` which allows the basic of representation of agents by spheres in 3D space. The header file contains parameters, such as size of spheres, which are passed to OPEN GL source file (`"visualization.cu"`) and they are defined manually.

Chapter 4

Immune System

Immune system is a biological defense system in living organism that protects them from invasion of external pathogens such as bacteria, viruses, and fungi and keeps functional tissues healthy. All forms of living organisms are equipped with immune system and they could be primitive (immune system of single celled bacteria) or complex (immune system of human beings). It must possess an ability to evolve and adapt since the pathogens are increasingly evolving to avoid detection by the immune system.

The immune system protects the living organisms from infection using a layered defense mechanism. The first layer of this defense mechanism is a physical or external barriers ,such as skin, which prevents the entry of bacteria and viruses into the body. If it fails to prevent the entry of pathogens, the first line defense mechanism called innate immune system (59) acts upon these external pathogens in response to distressed signals from infected cells (1) and kills them (the cells) upon contact. It is non-specific and acts upon pathogens in a generic way (60). If the innate immune system fails to eliminate the pathogens completely, the second line of defense called adaptive immune system (60) is activated by innate immune system. The adaptive immune system is antigen specific and act upon pathogens that matches its specificity. Moreover, it evolves as it eliminates the pathogens and develops an immunological memory so that the immune response is faster and stronger when immune system is invaded by same pathogen (3) once again. The detail description of innate and adaptive immune system is described in the section below.

4.1 Innate immune system

The innate immune system is the first line defense (59) of the immune system against invasion by foreign entities. They are non- specific and act upon different invaders in a generic way. Besides, it is incapable of providing long lasting protection to the host as they lack immunological memory (60). The essential functions of innate immune system are: activate adaptive immune system by presenting antigens to adaptive immune cells, create a barrier (physical and chemical) to harmful pathogens, identify and remove pathogens from tissues, organs and blood by white blood cells, promote complement system (61) for removal of dead cells and respond to distressed signals from infected cells by recruiting innate immune cells. The components of innate immune system (Figure 4.1 (62)) is categorized based on its functionality and they are:

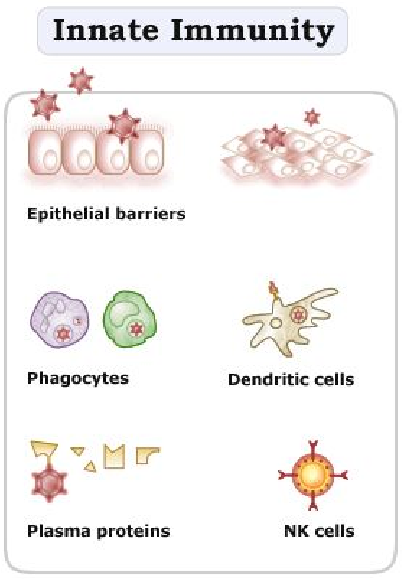


Figure 4.1: Components of innate immune system

4.1.1 Anatomical barrier

Epithelial tissues of skin, respiratory tract and gastrointestinal tract are the examples of anatomical barriers which prevents the penetration of foreign microbes by forming physical or chemical barriers. Integrity of cells in epithelial tissue form a physical barrier which serves as first line of defense to prevent the entry of pathogens (60). They also secrete anti-microbial chemicals that suppress the growth of microbes (62). Furthermore, respiratory and gastrointestinal tract also create a physical barrier by trapping pathogens in mucus produced by them (63). The elimination of microbes by stomach acids and digestive enzymes in a stomach is an example of chemical barrier created by gastrointestinal tract (62).

4.1.2 Complement system

Complement system, a part of innate immune system, is composed of more than twenty sets of protein molecules in the blood stream in an inactive state (61). They are activated when they encounter a site of infection where they attack the surface of pathogen. This leads to series of events that eliminate microbes, and therefore, infection. Upon activation, complement proteins may bind to antibodies that adhere to surface of microbes and form a biochemical cascade (64) to activate other complement proteins. The cascade produces peptides that attract immune cells to kill microbes (65). Besides, the proteins may also be bound to carbohydrates on the surface where it forms a layer of complements which disrupt the plasma membrane of microbes, thereby, killing it (66).

4.1.3 Innate immune cells

Innate immune cells are White Blood Cells (WBCs) that form a second line defense of innate immune system (60). They are independent, not the part of any specific tissues or organs, which function as single cell organisms capable of moving freely, identifying and eliminating foreign particles and harmful micro-organisms. They are product of stem cells present in bone marrow (60). They eliminate pathogens by direct contact or by engulfing them (61). Innate immune cells also act as a mediator to activate adaptive immune system (67). They can be classified based on their functionality and they are:

1. Phagocytes

Phagocytes are one of the group of innate immune cells that remove foreign particles, bacteria, dead or dying cells that are at the end of their life cycle (68). There are about six million phagocytes in one liter of human blood (69). The name phagocyte is derived from the Greek word, *phagein* and *cyte* which means "eater cells" (70). They simply ingest the microbes and neutralize them by a process called "Phagocytosis" (Figure 4.2) (71; 72).

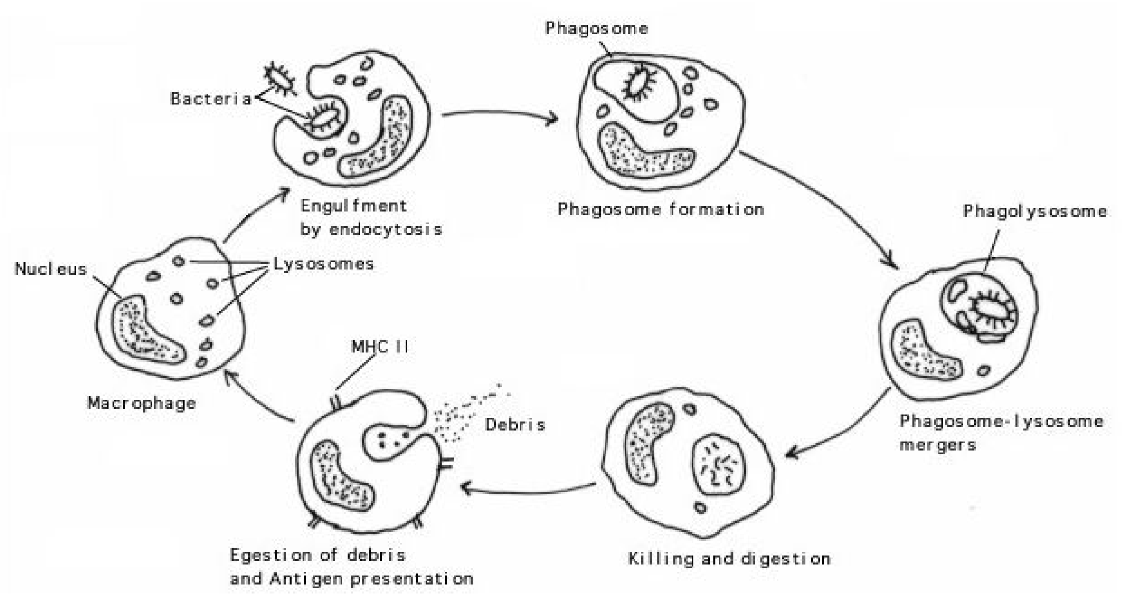


Figure 4.2: Phagocytosis by a Macrophage

In the process of phagocyotsis, the phagocytes extend the portion of their plasma membrane to engulf the microbes until they are completely enclose inside their membrane (61). They produce digestive enzymes to kill microbes followed by respiratory burst with a release of free radicals (73; 74). Phagocytes are either "professional" or "non-professional" based on effectiveness of phagocytosis they perform (71). The professional phagocytes have receptors on their surface that is capable of identifying micro-organisms that are not the part of the body (71). Besides, they also possess a capability to follow distressed signals produced by infected cells by a process called chemotaxis. After the completion of phagocytosis, the cells such as macrophages and dendritic cells extract the antigen from microbes and present it to adaptive immune cells. On the other hand, non-professional phagocytes are responsible for scavenging dead cells and create suitable conditions for regeneration of healthy cells (75). The different type of phagocytes are described below:

a. Macrophages

The name Macrophage (MΦ) comes from the Greek word *makros* and *phagein* which means big eaters (70). A typical size of a macrophage found in human body is 21 μm (76) with an average life span ranging from 4 to 15 days (71).

They are usually found residing in the tissues where they are activated when they sense the presence of foreign particles (77). They ingest and neutralize foreign particles, bacteria and cancer cells by phagocytosis. There are two types of macrophages: M1s that promote inflammation which is result of response to infection or injury and M2s that reduce inflammation (anti-inflammatory) and promote the repair of damaged tissues by scavenging the dead or dying cells (78).

Apart from inflammatory and anti-inflammatory response, they play an important role in activation of adaptive immune cells. After the microbe is neutralized by digestive enzymes in phagocytosis, they extract an antigen from it and attach it to a MHC (Major Histocompatibilty Complex) class II molecule (61), located on its surface, so that other white blood cells can distinguish them from pathogens.

They present an antigen to helper T cell for killing of pathogens with same antigen specificity (61).

b. Neutrophils

Neutrophils are the type of White Blood Cells (WBCs) that constitutes majority of innate immune cells (50% to 60% of WBCs) (79). The number of neutrophils present in one liter of human blood is approximately five billion (69). It has an average size of 10μm in diameter (80) with average life span of 5 days (81). The neutrophils circulating in the blood stream are activated by danger signals and enter the site of infection from blood stream where they kill microbes and infected cells coated with antigen and complement proteins, thereby, causing inflammation (81). After the kill, neutrophils die and are released as pus (81).

c. Dendritic Cells

Unlike the macrophages which kills and scavenge infected cells and microbes, dendritic cells plays an important role in tissue surveillance to monitor the well being of cells in tissues. They are usually present in tissues that are in contact with external environment such as skin as well as in the stomach, lungs and inner portion of nose (69).

Dendritic cells are equipped with a special protrusion called dendrites (82) to engulf microbes and perform antigen presentation for activation of T cells and B cells (83; 81). After they gather antigen from pathogens, they are activated and travel to lymphoid tissues to initiate adaptive immune response by presenting antigen to T and B cells (84). The presentation of antigen activates T cells which later differentiate to helper T cells and killer T cells to kill pathogens that matches its antigen specificity (85). The helper T cells prepare B cells to produce antibodies specific to antigen presented.

2. Mast cells

Mast cells are another group of White Blood Cells (WBCs) derived from myeloid stem cell that trigger an immune response (inflammatory) leading to allergies (86) (79). They are similar to basophils in terms of outlook and their functions and differ from basophils by their location of residence. Mast cells usually are found in tissues (eg: mucosal tissues) whereas basophils are present in blood stream (87).

3. Basophils and Eosinophils

Basophils and Eosinophils, often termed as Basophil granulocytes and Eosinophil granulocytes, are innate immune cells that kill parasites by releasing poisonous toxins. Basophils are activated when they confront pathogens and release a chemical called histamine (61) to eliminate them. The killing of parasites by such chemical lead to hypersensitivity and allergic reactions such as asthma (79). Like basophils, eosinophils also release toxins and free radicals upon activation by parasites and bacteria, thereby, killing them instantly. However, these toxins also causes the destruction of healthy cells causing tissue damage which ultimately leads to allergic reactions (79).

4. Natural Killers

Natural killers (NKs) are the group of innate immune cells that kill the cells infected by viruses in response to danger signals emitted by infected cells. They mature and proliferate in bone marrow, spleen, lymph nodes and thymus and enter the blood stream for circulation (88). Unlike the other immune cells which require detection of antigen attached to the surface of infected cells, natural killers possess an ability to identify infected cells without the presence of antibodies and antigens (61). This feature of NKs produces faster immune response as they can detect harmful cells without an antigen on its surface which cannot be identified by other immune cells(89).

4.2 Adaptive immune system

Adaptive immune system consists of highly specialized antigen specific cells that eliminate pathogens matching their antigen specificity and develop an immunological memory to provide an effective protection to the body for long period of time when infested by the same group of pathogens. For example, a patient who has recovered from measles will be immune to it in the future. It is also referred to as acquired immunity since it is developed during the life time due to the production of antibodies when exposed to an antigen (90).

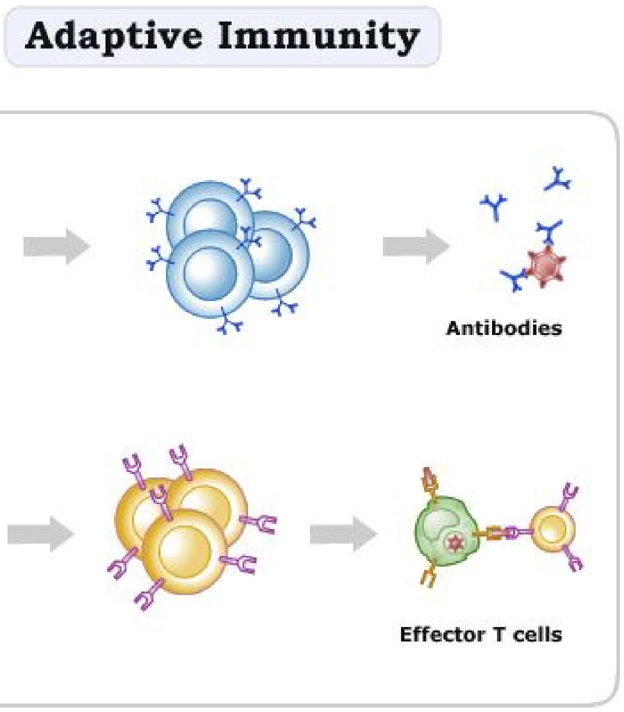


Figure 4.3: Components of adaptive immune system

The adaptive immune system consists of group of White Blood Cells (WBCs) called lymphocytes (B cells and T cells) and is driven by two kinds of responses: antibody and cell mediated response. The antibody response is triggered by activation of B cells producing antibodies. Antibodies are made up of proteins called immunoglobulins that bind to an antigen, thereby, inactivating it to prevent infection of host cells (60). Cell mediated response, on the other hand, is driven by activation of T cells which seeks for antigen bound cells and pathogens and kill them upon contact. The components of adaptive immune system is shown in Figure 4.3 (62) and are described in the section below:

4.2.1 T Lymphocytes (T cell)

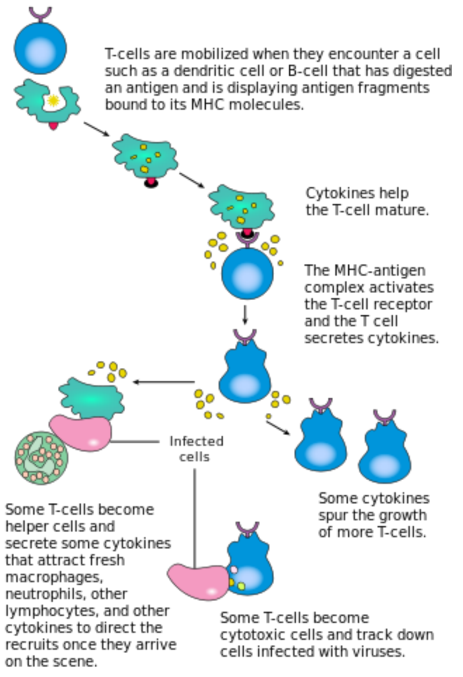

T-cells are mobilized when they encounter a cell such as a dendritic cell or B-cell that has digested an antigen and is displaying antigen fragments bound to its MHC molecules.
Cytokines help the T-cell mature.
The MHC-antigen complex activates the T-cell receptor and the T cell secretes cytokines.
Infected cells
Some cytokines spur the growth of more T-cells.
Some T-cells become helper cells and secrete some cytokines that attract fresh macrophages, neutrophils, other lymphocytes, and other cytokines to direct the recruits once they arrive on the scene.
Some T-cells become cytotoxic cells and track down cells infected with viruses.

Figure 4.4: The process involving T cell activation

T cells are the group of lymphocytes that drive cell mediated response of adaptive immune system to kill stressed cells or parasites matching its antigen specificity (91) as well as help B cells for their activation (91). They are produced in thymus where they mature and then enter the blood stream for circulation (60). They are equipped with T Cell Receptor (TCR) on their surface to detect antigen present on MHC class II molecule of dendritic cells and B cells (92).

T cells are in naive state during the start of their life cycle. They circulate in the blood stream after they leave the thymus. When they detect and make contact with

MHC class II molecules containing an antigen on the surface of B cells or dendritic cells, they get activated and start releasing chemicals called cytokines (93). After its activation, they may transition into helper, killer or memory cells as shown in Figure 4.4(93).

a. Helper T Cell

The primary function of helper T cell is to manage activities of other immune cells (marcophages and Bcells) by releasing cytokines rather than performing phagocytic or cyto-toxic activities by it self (91). It has a T Cell Receptor (TCR) that detects and recognizes MHC class II molecules present on the surface of Antigen Presenting Cell (APC). The recognition of MHC class II molecules activates naive helper T cell and activation of helper T cell results in release of cytokines which attract macrophages to the site of infection (94). It also activates Antigen Presenting Cells (APCs) that present an antigen to helper T cell.

Helper T cells are of two types, Th1 and Th2, and are classified based on the contact they make with particular type of Antigen Presenting Cells (APCs) (91). Helper T cell of type Th1 drives cell mediated immunity by influencing macrophages to kill stressed cells (91). On the other hand, helper T cell of type Th2 activates B cells to produce non-cytolytic antibodies (91). Th1 helper T cells are effective against intracellular microbes such as viruses which reside inside the host cells where as Th2 helper T cells are more powerful against extracellular parasites such as bacteria and toxins (91).

b. Cytotoxic and Memory T Cell

Cytotoxic T cell or Killer T Cell is one of the forms of T cell that causes the death of infected or damaged cells by releasing cytotoxins (91). The naive T cells transition to Cytotoxic T cells when T Cell Receptor (TCR) detect peptide bound MHC class

I molecules which activates killer T cells (91). Upon activation, killer T cells proliferate exponentially and seek out other peptide bound MHC class I molecules. The cytotoxins released by Cytotoxic T Cell perforates the plasma membrane of infected cells through which water and ions enter into the cytoplasm. This causes an infected cell to explode (91). The killer T cells die after the infection is cleared, however, few of them transition to memory T cells which develop an immunological memory to a specific antigen (94). If the same group of pathogens invade the system in the future, these memory cells transition back to killer cells, hence, resulting in faster immune response.

4.2.2 B Lymphocytes (B cell)

B cells are Antigen Presenting Cells (APCs) that drives the humoral response of the immune system by producing antibodies when they encounter antigens that matches their specificity. They are produced in bone marrow of mammals (60) and begin as immature B cells that enter into the blood stream where they transition into plasma cells that produce antibodies when they sense antigens and cytokines released by T cells (91). They have a receptor similar to T cells called B Cell Receptor (BCR) which binds a specific antigen. The difference between receptor of B and T cell is that BCR recognizes an antigen in unprocessed form whereas TCR can only detect processed antigens bounded with peptide (91).

The immature B cells circulating in the lymphatic system remain inactive and cannot produce antibodies until they encounter antigens of specific type in the blood stream. When they detect an antigen in its native form, the BCR present on their surface binds the unprocessed antigen. The unprocessed antigen bound on BCR is engulfed within plasma membrane of B cell where it is processed (93). The processed antigen is bound to its MHC molecule in the BCR receptor to present it to other immune cells such T cells. B cells activate and proliferate when they make contact with T cells in presence of cytokines released by T cells and transition to either short-lived plasma cells or memory B cells. The process of activation of B cells is shown in Figure4.5 (93).

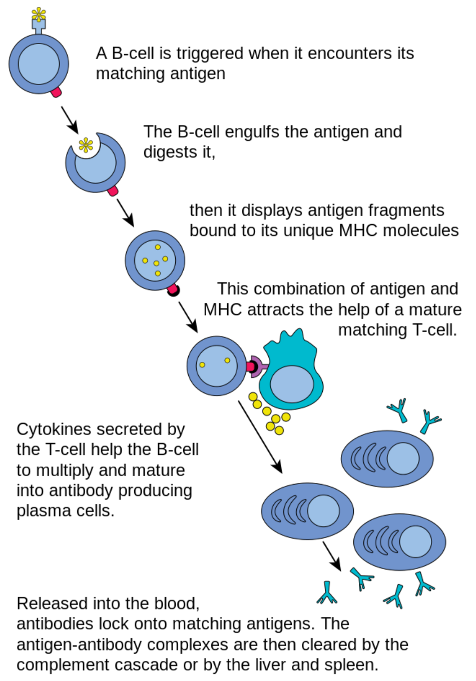

A B-cell is triggered when it encounters its matching antigen
The B-cell engulfs the antigen and digests it,
then it displays antigen fragments bound to its unique MHC molecules
This combination of antigen and MHC attracts the help of a mature matching T-cell.
Cytokines secreted by the T-cell help the B-cell to multiply and mature into antibody producing plasma cells.
Released into the blood, antibodies lock onto matching antigens. The antigen-antibody complexes are then cleared by the complement cascade or by the liver and spleen.

Figure 4.5: The process involving B cell activation

Each Plasma B cell is responsible for producing hundreds and thousands of antibodies (92) whose primary function is to activate complement proteins and help in elimination of antigens by binding and presenting them to phagocytes or killer immune cells (91). Plasma cells undergo apoptosis once the body recovers from infection. However, few plasma cells (about 10%) transition to memory B cells that develop an immunological memory against infection (91).

Chapter 5

Implementation of Immune Simulator in FLAME GPU

Unlike the traditional framework for ABMs such as Repast which is based on algorithms that are executed serially on CPUs, FLAME GPU utilizes the parallel computational feature of GPUs, thereby significantly increasing computational performance. This results in efficient simulation of even large models that cannot be handled with frameworks such as Repast. Additionally, visualization is relatively inexpensive to achieve since agent data is located in GPU memory where it can be rendered directly without any additional computational overhead.

5.1 Definition of immune cells in FLAME GPU

As discussed in chapter 2, different immune cells in the immune simulator can be represented as agents executing their own set of behavioral rules. The agent and message variables associated with immune cells, agent transition function definition handling their state changes and layers which schedule the execution of agent functions are defined within XMML model file definition. An example of specification for one of the cell type (parenchymal cell) is shown in Figure 5.1. As shown in the Figure 5.1, `memory` element in XMML model file consists of definition of agent variables and their type (`int`, `float`, `double`) associated with a parenchymal cell that store its position coordinates, state, and amount of chemical signals emitted when infected by a virus. The initial value of variables (Figure 5.2a) are defined in a file with `xml` extension and these values are supplied as input into simulation.

```
<xagents>
  <gpu:xagent>
   <name>Parenchymal_cell</name>
    <memory>
     <gpu:variable><type>int</type><name>id</name>
     </gpu:variable>
     <gpu:variable><type>float</type><name>x</name>
     </gpu:variable>
     <gpu:variable><type>float</type><name>y</name>
     </gpu:variable>
     <gpu:variable><type>float</type><name>z</name>
     </gpu:variable>
     <gpu:variable><type>int</type><name>state</name>
     </gpu:variable>
     <gpu:variable><type>int</type><name>bearAb</name>
     </gpu:variable>
     <gpu:variable><type>float</type><name>s_PK1</name>
     </gpu:variable>
     <gpu:variable><type>float</type><name>s_virus</name>
     </gpu:variable>
     <gpu:variable><type>float</type><name>s_apoptic</name>
     </gpu:variable>
     <gpu:variable><type>float</type><name>s_necrotic</name>
     </gpu:variable>
     <gpu:variable><type>int</type><name>stressedTime</name>
     </gpu:variable>
     <gpu:variable><type>int</type><name>killDelay</name>
     </gpu:variable>
    <functions>...</functions>
    <states>
      <gpu:state>
       <name>default_pcells</name>
       </gpu:state>
      <initialState>default_pcells</initialState>
    </states>
    <gpu:type>continuous</gpu:type>
    <gpu:bufferSize>5457</gpu:bufferSize>
  </gpu:xagent>
 </xagents>
```

Figure 5.1: XMML model file definition for Parenchymal cells

```
<xagent>
   <name>Parenchymal_cell</name>
   <id>0</id>
   <x>−1</x>
   <y>−1</y>
   <z>0</z>
   <state>4</state>
   <bearAb>0</bearAb>
   <s_PK1>0</s_PK1>
   <s_virus>0</s_virus>
   <s_apoptic>0</s_apoptic>
   <s_necrotic>0</s_necrotic>
   <stressedTime>0</stressedTime>
</xagent>
```

```
  <xagents>
    <gpu:xagent>
      <name>Parenchymal_cell</name>
      <memory> . . . </memory>
      <functions>
        <gpu:function>
          <name>state_pcells</name>
          <currentState>default_pcells</currentState>
          <nextState>default_pcells</nextState>
          <inputs>
            <gpu:input>
              <messageName>agent_bcells</messageName>
            </gpu:input>
          </inputs>
          <outputs>
            <gpu:output>
              <messageName>agent_pcells</messageName>
              <gpu:type>single_message</gpu:type>
            </gpu:output>
          </outputs>
          <gpu:reallocate>false</gpu:reallocate>
          <gpu:RNG>false</gpu:RNG>
        </gpu:function>
      </functions>
    </gpu:xagent>
   </xagents>
```

Figure 5.2: a) Figure on the top shows initial value of variables for one Parenchymal cell b) Figure on the bottom is an example of agent function definition (`state_pcells`)

Additionally, `function` element includes the definition of all the agent transition functions associated with the parenchymal cell that handle its state change, and input/output of messages. An example that illustrates the definition of one of the agent functions of parenchymal cell i.e. `state_pcells` is shown in Figure 5.2b. The message names i.e. `agent_bcells` and `agent_pcells` defined within `inputs` and `outputs` element inside `state_pcells` function, create a template for message API functions to read and write messages from message board. Besides, the function `state_pcells` also includes C based script that determines whether parenchymal cells are healthy or infected by a virus.

5.2 Simulation domain

The simulation domain implemented in FLAME GPU is adapted from previous version of Basic Immune Simulator (BIS) developed in RePast to preserve qualitative nature of the simulation. The simulation domain is divided into three zones namely: Zone 1, Zone 2 and Zone 3 (6; 7). Figure 5.3 shows the virtual representation of three different zones of the immune simulator implemented in FLAME GPU which is similar to its RePast implementation. Zone 1 is a representation of site for viral infection of generic parenchymal cells. The healthy parenchymal cells are represented by yellow spheres and the infected parenchymal cells are represented by green spheres as shown in Figure 5.3a. Zone 1 is also a residence for tissue surveilling innate immune cells called Dendritic cells (DCs). Zone 2 (Figure 5.3b) is a residence for adaptive immune cells (B cell agents, T cell agents and Cytotoxic T Lymphocyte agents) and represents lymph nodes or spleen. Zone 3 (Figure 5.3c) is a representation of blood circulation system which acts as a media to supply adaptive immune cells to site of viral infection i.e. Zone 1.

Adaptive immune cells move randomly in Zone 3 for some period of time before

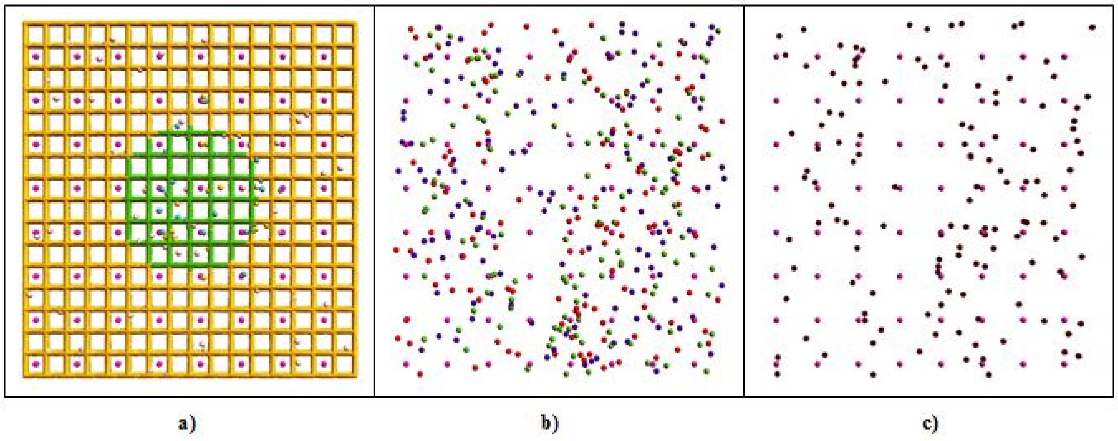


Figure 5.3: Virtual representation of three different zone in FLAME GPU: a) Zone 1: Site for infection of parenchymal cells by a virus b) Zone 2: Representation of lymph node or spleen c) Zone 3: Representation of blood circulation system

they move to Zone 1. Zone 3 also contains granulocyte agents which are the type of white blood cells for fighting the infection. Each zone contains portal agents which represent lymphatic and blood vessels. These portal agents are responsible for transferring immune cells from one zone to the other.

5.3 Agent types

The agents that take part in immune system simulation are generic functional tissue cells (Parenchymal cells), innate immune cells (Macrophages, Natural Killers, Dendritic cells and Granulocytes) and adaptive immune cells (B cells, T cells and CTLs). Innate immune cells originate in bone marrow (95). These immune cells are the first ones to respond to distressed signals (1) produced by infected parenchymal cells. They enter Zone 1 via portals present in this zone.

Natural Killers kill infected parenchymal cells. Macrophages are responsible for killing infected parenchymal cells as well as scavenging of dead parenchymal cells. The process of scavenging of dead parenchymal cells sets up a necessary condition for regeneration of healthy parenchymal cells. Dendritic cells gather antigens from infected parenchymal cells and travel to Zone 2 i.e. the lymph nodes, through portals and present antigens to adaptive immune cells that matches its specificity. Activation, proliferation, as well as transition of adaptive immune cells (B cells, T cells, CTLs) to memory cells take place in Zone 2 when they come in contact with dendritic cells or with each other. The activated adaptive immune cells first move to Zone 3 where they randomly move for some time and finally travel to Zone 1.

T cells represent helper T cells which manage activities of macrophages and B cells by releasing cytokines (91). CTLs are killer cells responsible for killing infected parenchymal cells that match their antigen specificity. B cells, on the other hand, produce antibodies against the antigens making them primary targets for killer immune cells. Agent motion is influenced by chemical signals detected in the proximal environment. Immune cells would tend to move randomly if there isn't any chemical signal around them. If they detect any chemical signal in their vicinity, they will follow the signal with highest concentration (96).The rules for the state change of different type of cells participating in simulation is described in the sub-section below:

5.3.1 Parenchymal cell agents

Parenchymal cells are the generic functional tissue cells that remain stationary throughout the course of simulation. These agents are initialized in a HEALTHY state during the start of the simulation. The viral infection starts at the center of Zone 1 which gradually spreads out infecting all neighboring parenchymal cells. The infected parenchymal cells release a distressed signal, parenchymal-kine 1 (PK1) in the form of heat shock proteins (97), uric acid (98) or chemerin (99).

The infected PCs then make a state transition to one of three states: STRESSED, CHALLENGED or TARGETED. The challenged PCs are target for natural killers, pro-inflammatory T cells or CTLs and are killed upon contact. They may undergo lysis in presence of complement products(C') and antibody Ab1 (100). The challenged PCs make a transition to state TARGETED when they are bounded by antibody produced by B cells making them susceptible for killing by pro-inflammatory macrophage agents (101). This causes neighboring parenchymal cells to become stressed as a result of reactive oxygen species released by pro-inflammatory macrophages (102). The granulocytes also act upon the stressed PCs and kill them by a release of lethal degranulation product 1 (G1) (102; 103). The dead parenchymal cells are scavenged by macrophages facilitating necessary conditions for regeneration of healthy PCs (104).

Algorithm 1 State change : Parenchymal Cells

```
1: procedure STATE_PARENCHYMAL_CELLS()
2:     if (state==HEALTHY & virus>Ab1+Ab2) then
3:         state=CHALLENGED
4:     end if
5:     if (state==CHALLENGED) then
6:         PK1=outputSignal
7:         if (Ab1+Ab2>virus & (bearAb==FALSE) & (Ab2≥0) then
8:             state=TARGETED
9:         end if
10:        if (NK||Ts||MΦs ) then
11:            state=APOPTIC
12:        end if
13:    end if
14:    if (C'>0) & ((Ab1-(virus+Ab2)>Ab_Lysis) then
15:        state=NECROTIC
16:    end if
17:    if (state==STRESSTED) then
18:        PK1=outputSignal
19:        if (life>stressedTime+DURATION_STRESSED) then
20:            state=HEALTHY
21:        end if
22:    end if
```

5.3.2 Dendritic cell agents

The DCs begin in an INACTIVE state in zone 1 and are of two types: pro-inflammatory (DC1) and anti-inflammatory (DC2). The DCs in INACTIVE state can transition to two possible states: ACTIVATED or AG-PRIMED (antigen primed) depending upon the type of signal it detects.

Algorithm 2 State change : Dendritic Cells - Zone 1

```
procedure STATE_DENDRITIC_CELLS_ZONE1()
    if (life < LIFE_DC_ZONE1) then
        if (state==INACTIVE) then
            if (PK1>0) then
                state=ACTIVATED
                if (MK1≫MK2 & type==DC1) then
                    type=DC2
                end if
            end if
            if (virus>200) then
                state = AG-PRIMED
                if (Ab1>200 & Ab2>200 & type==DC1) then
                    type=DC2
                end if
                if (CK1>200) & type==DC2 then
                    type=DC1
                end if
            end if
        end if
        if (state==ACTIVATED) then
            if (virus || CHALLENGED PC contact) then
                state=AG-PRIMED
            end if
        end if
    end if
    if (life>LIFE_DC_ZONE1) then
        state=APOPTIC
    end if
```

The dendritic cells of type DC1 or DC2 move randomly until it detects shock signal (PK1) produced by infected parenchymal cells. The detection of PK1 in its immediate environment causes DCs to change to ACTIVATED state (105). The activated DCs produce a chemical signal MK1 (Mono-kine 1) or MK2 (Mono-kine 2) depending on its type. For example, INACTIVE DC1 changes to ACTIVATED DC1 and INACTIVE DC2 changes to ACTIVATED DC2 upon detection of PK1.

However, if signal MK2 (Mono-kine 2) is greater than signal MK1 (Mono-kine 1), it converts to ACTIVATED DC2 (106).

Algorithm 3 State change : Dendritic Cells - Zone 2

```
procedure STATE_DENDRITIC_CELLS_ZONE2()
    if (life < LIFE_DC_ZONE1 & zone==2) then
        if (state==AG-PRIMED) then
            if (type==DC1 || type==DC2) then
                if (timerMK<DURATION_MK_ZONE2) then
                    MK1 or MK2 = outputSignal
                    timerMK1 or timerMK2 += 1
                end if
                if (Ag-matched B1 or B2) then
                    life=0
                end if
                if (Ag-matched T1 or T2) then
                    MK1 or MK2 = 0
                    NumTsContact += 1
                end if
            end if
        end if
    end if
    if (numTsContact≥12) then
        state=APOPTIC
    end if
    if (life>LIFE_DC_ZONE1) then
        state=APOPTIC
    end if
```

The DCs attain antigen-primed (AG-PRIMED) state under three conditions. First, the detection of virus signal in the proximal location of DC1 or DC2 causes them to convert to AG-PRIMED DC1 or AG-PRIMED DC2 (107). Second, the presence of virus along with antibody changes INACTIVE DC1 to AG-PRIMED DC2 (108). INACTIVE DC2 is transitioned to AG-PRIMED DC1 in presence of virus and signal CK1 (cyto-kine 1) (109). Third, the DCs in ACTIVATED state when make contact with virus infected PCs change to AG-PRIMED state respective of its type (110). The AG-PRIMED DCs move to Zone 2 via portals present in Zone 1 to present the antigen to adaptive immune cells (B cells, T cells and CTLs) (110).The

DCs undergo apoptosis if they fail to detect any infected PCs or stress signal within their lifetime (111).

After the DCs move to Zone 2, they move randomly for some time and become stationary. They continue to produce MK1 or MK2 in Zone 2 as well. At this moment, DCs wait for antigen matched adaptive immune cells (B cells, T cells and CTLs) to make contact with them. The contact with DCs affects the state of adaptive immune cells as well as their own state. For example, contact with antigen matched B cells extends the life of DCs (112) and contact with antigen matched T cells resets the production of MK1 or MK2. The DCs undergo apoptosis if they reach the allocated life time in Zone 2 or exceed threshold for T cells contact (113; 114).

5.3.3 Macrophage agents

The macrophages (MΦs) enter Zone 1 in naive state (MΦ0) when portals present in Zone 1 sense PK1 emitted by infected PCs. The state change of MΦs is determined by type of signals it senses from its local environment.

The presence of PK1, CK1, C' (complement products) and necrotic debris (100) cause MΦs to transition to ACTIVATED MΦ1s (pro-inflammatory) whereas sensing of apoptic signal and antigen-antibody (Ag-Ab) complexes (101) cause MΦs to change to ACTIVATED MΦ2s (anti-inflammatory). Both MΦ1 and MΦ2 in activated state have the ability to scavenge dead PCs which provides necessary condition for regeneration of healthy PCs. MΦ1 also has the ability to kill infected PCs. MΦ1 and MΦ2 in activated state produce signals MK1 and MK2 respectively. The killing of infected PCs and scavenging of dead PCs cause MΦs to attain antigen-primed (AG-PRIMED) state respective of their type. The MΦs in AG-PRIMED state posses ability from previous state to scavenge dead PCs as well as kill PCs bound by a antibody. If MΦs in this state make contact with antigen-matched T cells, it extends the life of MΦs. The MΦs in activated and antigen primed state undergo apoptosis when their life time is exhausted.

Algorithm 4 State change : Macrophages

```
procedure STATE_MACROPHAGES()
    if (life < LIFE_MΦ_ZONE1) then
        if (state==MΦ0) then
            if (PK1>0 ‖ CK1>0 ‖ C'>0 ‖ Nec. debris>0 ) then
                state=ACTIVATED MΦ1
            end if
            if Apop.Signal>0 ‖ Ab-Ag complexes then
                state=ACTIVATED MΦ2
            end if
        end if
        if (state==ACTIV. MΦ1 ‖ ACTIV. MΦ2) then
            if (kill PCs ‖ scavenge PCs) then
                state = AG-PRIMED MΦ1 or MΦ2
            end if
        end if
    end if
    if (life>LIFE_MΦ_ZONE1) then
        state=APOPTIC
    end if
```

5.3.4 Natural Killer agents

Natural Killers (NKs) are introduced to Zone 1 when portals in Zone 1 sense PK1 from PCs. They move randomly for some time. When they detect PK1 that is being emitted from infected PCs, they follow PK1 with the highest concentration eventually seeking infected PCs. If the signal PK1 is greater than CK1 around the Natural Killer of interest, killing of infected PCs takes place upon contact. At this time, release of chemical signal CK1 by NKs also takes place (115). After the killing of infected PCs and CK1 production, NKs return to the state where they continue to move randomly. They have a limited number of kills and lifetime, after which they undergo apoptosis.

Algorithm 5 State change : Natural Killers

```
procedure STATE_NATURAL_KILLERS()
    if (life < LIFE_NK_ZONE1) then
        if (PK1>0) then
            followPK1=TRUE
        end if
        if (PK1>CK2 & followPK1==TRUE) then
            killPC = TRUE
        end if
        if (killPC==TRUE) then
            killCount+=1 ; CK1Timer=DURATION_NK_CK1
        end if
        if (CK1Timer>0) then
            CK1=outputSignal ; CK1Timer -= 1
        end if
        if (CK1Timer==0) then
            followPK1 = FALSE ; randomMotion = TRUE
        end if
        if (killCount ≥ NK_KILL_LIMIT) then
            state=APOPTIC
        end if
    end if
    if (life > LIFE_NK_ZONE1) then
        state=APOPTIC
    end if
```

5.3.5 Granulocyte cell agents

Granulocytes begin in Zone 3 where they are moving randomly. They travel to Zone 1 when portals present in Zone 3 sense presence of complement products (100) or MK1. After enter Zone 1, they move randomly until they detect complement products around them and eventually follows signal of highest concentration. They release degranulation product which is capable of killing any stressed PCs (103). They undergo apoptosis after their time is up.

Algorithm 6 State change : Granulocytes - Zone 1

```
procedure STATE_GRANULOCYTES_ZONE1()
    if (life < LIFE_GRAN_ZONE1) then
        G1=outputSignal
    end if
    if (life > LIFE_GRAN_ZONE1) then
        state=APOPTIC
    end if
```

5.3.6 T cell agents

T cells begin in Zone 2 i.e. lymph nodes in an INACTIVE state where they move randomly until it finds a DC matching its antigen specificity. The contact with DC1 or DC2 results in activation and proliferation of T1 or T2 respectively (116; 110; 113; 117) and release of signal CK1 or CK2.

Algorithm 7 State change : T cells - Zone2

```
procedure STATE_TCELLS_ZONE2()
    if (state==INACTIVE T0) then
        if (Ag-matched DC1 or DC2) then
            state=ACTIVATED T1 or T2
        end if
    end if
    if (state==ACTIVATED T1 or T2) then
        if timerCK < DURATION_CK_ZONE2 then
            CK1 or CK2 = outputSignal
            timerCK1 or timerCK2 += 1
        end if
        if (Ag-matched DC1 or DC2) then
            life += ADD_LIFE
            Birth_Ts = ADD_Ts
        end if
        if MK1==0 || CK1 == 0 then
            state = MEMORY T1 or T2
        end if
    end if
    if (state==ACTIVATED & life > LIFE_Ts_ZONE2) then
        state=APOPTIC
    end if
```

Additionally, contact with antigen matched B cells as well as series of contacts with DCs extend the life of Ts. ACTIVATED Ts make a transition to MEMORY Ts if there is an absence of CK1 or CK2 in the environment. However, contact with antigen matched DC makes MEMORY Ts to be ACTIVATED again. Ts move to Zone 3 where they randomly for some time before traveling to Zone 1. If ACTIVATED Ts fail to make contact with DC within certain period of time, they undergo apoptosis.

Algorithm 8 State change : T cells - Zone1

```
procedure STATE_TCELLS_ZONE1()
    if (life < LIFE_Ts_ZONE1) then
        if (state==ACTIVE_Ts||state==MEMORY_Ts) then
            if (Ag-matched MΦ contact) then
                CK1 or CK2 = outputSignal
            end if
            if (CHALLENGED PC) then
                killPC = TRUE ; killCount += 1
            end if
            if (killCount≥MAX_T_KILLS) then
                state=APOPTIC
            end if
        end if
    end if
    if (life>LIFE_Ts_ZONE1 then
        state=APOPTIC
    end if
```

T cells (Ts) enter Zone 1 in response to detection of PK1 signal by portals present in Zone 1. T cells of type T1 seek infected PCs by following PK1 with strongest concentration and kill them upon contact. If T1 or T2 makes contact with antigen-matched macrophage (MΦ), they start to produce CK1 or CK2 depending upon type of macrophage (MΦ) (118). T cells of type T1 undergo apoptosis if they reach beyond the threshold for number of kills (119) or exceed allocated lifetime in Zone 1.

5.3.7 CTL agents

Like other adaptive immune cells, CTLs begin in an INACTIVE state in zone 2, moving randomly waiting to make contact with an antigen-matched DC1. The contact with DC1 makes them ACTIVATED.

Algorithm 9 State change : CTLs - Zone2

```
procedure STATE_CTL_ZONE2()
    if (Ag-matched DC1) then
        if (state==INACTIVE) then
            state = ACTIVATED
        end if
        if (state==ACTIVATED  &  CK1==0) then
            state = MEMORY_CTLs
        end if
        if (state==MEMORY_CTLs  &  CK1>0) then
            state = ACTIVATED
        end if
    end if
    if (life < LIFE_CTLs_ZONE2) then
        if (state==ACTIVATED) then
            CK1 = outputSignal
            BIRTH_CTLs = ADD_CTLs
        end if
    end if
    if (life > LIFE_CTLs_ZONE2) then
        state=APOPTIC
    end if
```

In an ACTIVATED state they proliferate and release CK1 signal. ACTIVATED CTLs transition to MEMORY CTLs if they make contact with a DC1 in absence of CK1 signal (120). The presence of CK1 signal and contact with DC1 extends the life of CTLs. ACTIVATED CTLs then migrate to Zone 3 and eventually to Zone 1 where they kill infected PCs. CTLs enter Zone 1 in an activated state and continue to produce CK1 signal for a finite period of time.

Like T cells (Ts), they seek virus infected PCs by following PK1 emitted by such PCs and kill them upon contact. The contact with infected PCs also extend the duration of emission of CK1 signal by CTLs. ACTIVATED CTLs transition to MEMORY CTLs in absence of PK1 or CK1 in their immediate environment. The contact with infected PCs cause MEMORY CTLs to be ACTIVATED again. CTLs undergo apoptosis after their life is over.

Algorithm 10 State change : CTLs - Zone1

```
procedure STATE_CTL_ZONE1()
    if (CHALLENGED or TARGETED PC contact) then
        if (state==ACTIVATED) then
            killPC = TRUE
        end if
        if (state==MEMORY_CTLs) then
            state = ACTIVATED
        end if
    end if
    if (life<LIFE_CTL_ZONE1 & state==ACTIV.) then
        if (No CK1 or PK1) then
            state = MEMORY_CTLs
        end if
        if (ticker<DURATION_CK1_ZONE1) then
            CK1 = outputSignal ; ticker += 1 end if
    end if
    if (life > LIFE_CTL_ZONE1) then
        state=APOPTIC
    end if
```

5.3.8 B cell agents

B cells (Bs) reside in Zone 2 in an INACTIVE state waiting for antigen matched DCs to make contact with them (121). If they make contact with DC1, B cells of type B1 are produced and type B2 is the result of contact with DCs of type DC2. B cells of type (B1 or B2) gets ACTIVATED when they make contact with activated, antigen-matched T1 or T2. Bs in ACTIVATED state may transition into either of two states: GERMINAL or PLASMA.

Germinal cells in activated state produce anti-bodies and remain in Zone 2 where as Plasma B cells, on the other hand, travel to Zone 1 and produce anti-bodies there. Contact with Ts also leads to birth of B cells. ACTIVATED Bs make a transition to MEMORY Bs when they make series of contacts with antigen matched DC1 or DC2, thus, extending life of B cells. Memory Bs are again activated when they make contact with antigen matched Ts in presence of cytokines (MK1, CK1 or MK2, CK2).

Algorithm 11 State change : B cells - Zone2

```
procedure STATE_BCELLS_ZONE2()
    if (Ag-matched DC1 or DC2) then
        if (type==B1 or B2 & state==ACTIV.) then
            DCcontacts + =1 ; life += ADD_LIFE
        end if
        if (type==B1 or B2 & state==MEM_Bs) then
            DCcontacts + =1
        end if
        if (type==B0) then ; type = B1 or B2
        end if
    end if
    if (Ag-matched T1 or T2) then
        if (state==INACTIVE) then
            if (type==B0) then
                type = B1 or B2
            end if
            if (type==B1 or B2) then
                state=ACTIVATED ; flag=rand() ; Birth_Bs = ADD_Bs
                mode=(flag>0.5) ? GERMINAL:PLASMA
            end if
        end if
        if (state==MEMORY_Bs) then
            state=ACTIVATED ; mode=PLASMA ; BIRTH_Bs=ADD_Bs
        end if
    end if
    if (state==ACTIVATED & DCcontacts$\geq$1 then
        if (mode==GERMINAL) then
            state = MEMORY_Bs ; life += ADD_LIFE
        end if
    end if
    if (state==MEMORY_Bs & DCcontacts$\geq$1) then
        life += ADD_LIFE
    end if
    if (state==ACTIVATED & type==B1 or B2) then
        if (AbTimer$\leq$DURATION_AB_ZONE2) then
            Ab1 or Ab2=outputSignal ; AbTimer += 1
        end if
    end if
    if (life > LIFE_B_ZONE1) then
        state=APOPTIC
    end if
```

ACTIVATED Bs undergo apoptosis after their life surpasses allocated lifetime.

Plasma B cells travel to Zone 1 from Zone 3 via portals present in Zone 1. They move randomly in Zone 1 where they produce antibodies in the presence of CK1 or CK2 for predefined period of time. If they don't detect CK1 or CK1 in its proximal location, they stop producing antibodies. They undergo apoptosis after their life is over.

Algorithm 12 State change : B cells - Zone1

```
 1: procedure STATE_BCELLS_ZONE1()
 2:     if (life < LIFE_Bs_ZONE1) then
 3:         if (type==B1 or B2 & CK1 or CK2>0) then
 4:             if (AbTicker < DURATION_AB_ZONE1) then
 5:                 Ab1 or Ab2 = outputSignal
 6:                 AbTicker += 1
 7:             end if
 8:         end if
 9:     end if
10:     if (life > LIFE_Bs_ZONE1) then
11:         state=APOPTIC
12:     end if
```

Chapter 6

Results

In this chapter, the results obtained from immune simulator implemented in FLAME GPU are compared with its RePast implementation to verify that immune cells executing in parallel demonstrate same behaviors, as in their RePast implementation, without loss in statistical accuracy. The test of statistical accuracy is important because there is possibility that results obtained from immune simulator implemented in FLAME GPU may deviate from its RePast version due to the difference in the way the random motion of immune cells are implemented. Furthermore, discrepancies in the order at which the state change of immune cells are executed could also lead to difference in results. For this reason, statistical comparison is required to ensure that results from immune simulator in FLAME GPU lie within statistical limits of results obtained from its RePast implementation. The measure of statistical accuracy was done by utilizing the statistical measures such as mean value and standard deviation. Apart from testing of statistical accuracy of immune simulator in FLAME GPU, a benchmark is carried out to demonstrate the performance advantage of using FLAME GPU when large population of immune cells execute their behavioral rules in the simulation.

6.1 Qualitative and statistical analysis

The qualitative analysis and validation of statistical accuracy for immune cells count at Zone 1 and Zone 2 were carried for immune win condition. Immune win is a situation in which infection is completely eradicated as result of elimination of challenged/stressed parenchymal cells by immune cells and sets up favorable conditions for regeneration of healthy parenchymal cells. The initial condition for immune win condition for immune simulator implemented in FLAME GPU was set, similar to as initial condition for immune simulator implemented in RePast, as shown in Table 6.1 (7).

Parameter	**Value**	**Parameter**	**Value**
Number of PCs	5457	Duration_Ab_Zone1	150
Viral_Infection_Threshold	50	NumTs_ToSend	2
Ab_Lysis_Threshold	100	Life_T_Zone1	20
Duration_Stressed	25	Life_T_Zone2	13
Number of DCs(Zone 1)	50	Life_T_Zone3	50
NumDC_ToSend	3	Duration_CK_Zone1/Zone2	25
Life_DC_Zone1	50	T_Max_Kills	10
Life_DC_Zone2	100	NumMΦ _ToSend	5
Duration_MK_Zone1/Zone2	25	Life_MΦ _Zone1	50
NumBs_ToSend	1	NumNK_ToSend	4
Life_B_Zone1	25	NK_Kill_Limit	15
Life_B_Zone2	10	NumCTL_ToSend	4
Life_B_Zone3	25	Life_CTL_Zone1/Zone2	25

Table 6.1: Initial values for immune win condition

The simulation was carried out to observe the behavior of immune cells in response to infection of parenchymal cells by a virus. The number of parenchymal cells that took part in the simulation was 5457. Out of 5457 PCs, 192 cells were initialized as cells infected by a virus and were placed in the space around the center region of Zone 1. As the simulation progresses, infection spreads out which causes healthy parenchymal cells, located near infected ones, to be stressed/challenged when virus signal in their local environment is greater than the threshold value (Virus_Infection_Threshold). The innate immune cells are the first group of immune cells to respond to the infection and enter Zone 1 when the portals present in Zone 1 sense PK1 signal emitted by infected cells. The average count of innate immune cells (Macrophages and Natural killers) in Zone 1, for both FLAME GPU and RePast implementation of immune simulator, is shown in Figure 6.3 which is obtained by running 50 trials of simulation for both FLAME GPU and RePast. Similarly, the average count of dendritic cells and adaptive immune cells (T cells, B cells and CTLs) in Zone 2 (Figure 6.2) was obtained in a same way as for innate immune cells count in Zone 1.

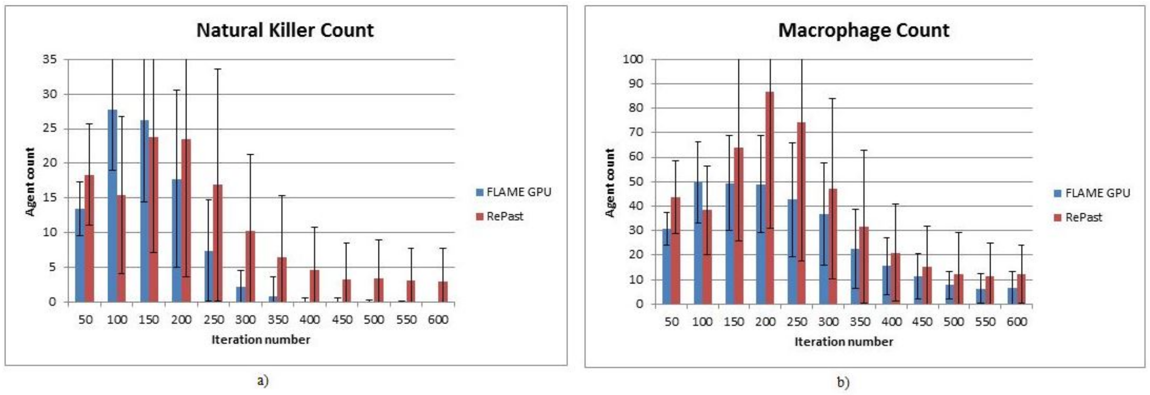


Figure 6.1: Innate immune cells count in Zone 1. a) count of Natural killers b) count of macrophages

The blue bars in Figure 6.3 represent the average count of innate immune cells (natural killers and macrophages) that are present in Zone 1 at different period of time (from infection to recovery). Similarly, the blue bars in Figure 6.2 shows the count of adaptive immune cells (T cells, B cells and CTLs) and dendritic cells in Zone 2. It is observed that the count of immune cells (both innate and adaptive) gradually increase as the number of parenchymal cells infected by a virus rises and eventually reaches a peak value when majority of parenchymal cells are infected. As the infected

PCs are cleared by immune cells, the count of immune cells also gradually decreases which indicate the process of recovery of parenchymal cells as dead PCs are replaced by healthy ones.

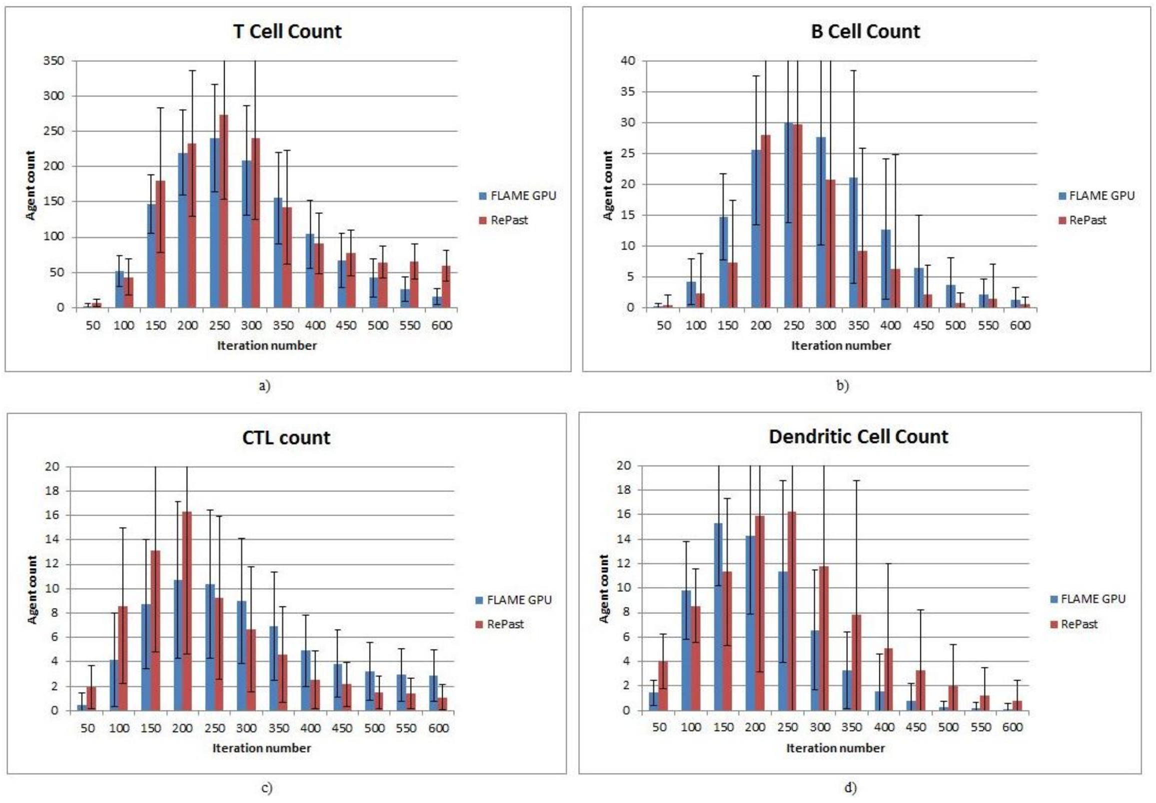


Figure 6.2: Count of different immune cells in Zone 2. a) count of T cells b) count of B cells c) count of CTLs d) count of dendritic cells

Furthermore, for statistical analysis the count of immune cells present in Zone 1 and Zone 2 of immune system implemented in FLAME GPU is compared with its RePast implementation. The red bars in Figures 6.3 and 6.2 shows the count of immune cells present in Zone 1 and Zone at different time steps. The standard deviation bars at each time step verify that count of immune cells in Zone 1 and Zone 2 of immune simulator developed with FLAME GPU lie within statistical limits of its RePast implementation. This outcome verifies its statistical accuracy i.e. parallelization of immune cell agents does not effect the quality of outcomes from immune simulator.

6.2 Benchmark

The benchmark demonstrates the performance advantage when the immune simulator developed with FLAME GPU simulates large population of immune cell agents in parallel. The benchmark was carried out by varying the initial immune cells count from 8,000 to 20,000. Since the number of parenchymal cells that take part in the simulation is fixed, the initial immune cells count was achieved by varying initial count of dendritic cells (DCs) from 500 to 3,500 as shown in Table 6.2 and keeping the count of other immune cells (B cell, T cell, NKs, MΦ and CTLs) unchanged.

Initial Agent count	**Initial DC count**	**Speed Up**
8,000	500	3.43
10,000	1,000	4.03
12,000	1,500	4.55
14,000	2,000	5.04
16,000	2,500	6.35
18,000	3,000	8.1
20,000	3,500	13.002

Table 6.2: Initial count of DCs for different agent population and simulation speed-up with FLAME GPU

The simulation was run in Intel core i7 2.67 GHz CPU with 6.00 GB RAM equipped with NVIDIA Tesla C2050 GPU on Windows 7 OS. The result of the benchmark was obtained by running 15 trials as the count of agent vary significantly in each trial due to stochastic nature of simulation. Figure 7 illustrates plot for speed-up obtained with FLAME GPU against agent population. It is observed that computational performance increased by 13 times when simulation was run for initial agent count was set to 20,000 agents.

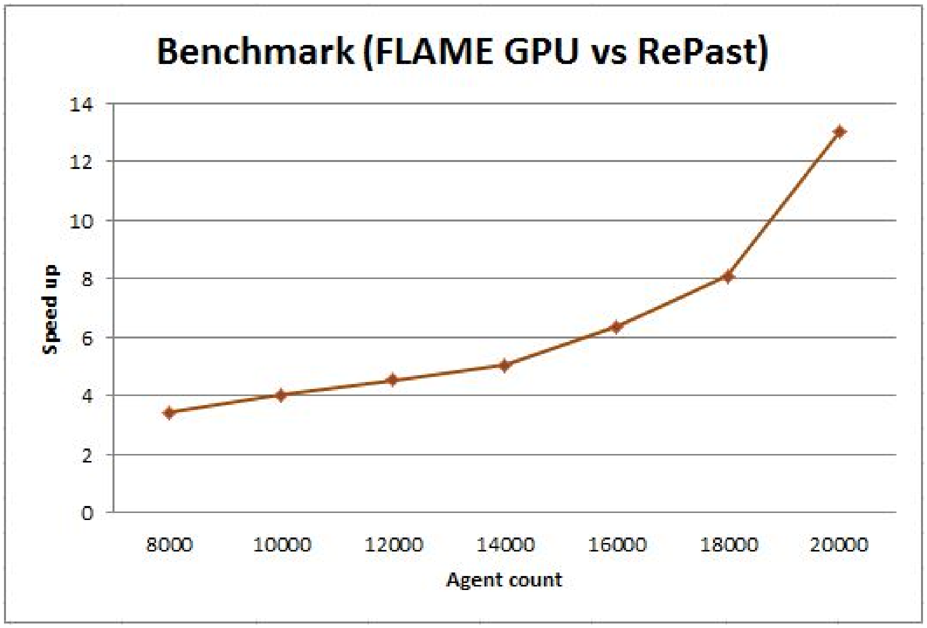


Figure 6.3: Benchmark: plot for speed-up obtained with FLAME GPU against agent count

However, performance analysis wasn't carried out for agent population greater than 20,000 as immune simulator implemented in RePast as it would slow down the simulation making it infeasible to operate. Hence, it can be noted that there is a significant improvement in computational performance.

Chapter 7

Conclusions and Future Work

The implementation of the basic immune simulator using FLAME GPU framework was successfully completed. The major contribution of this thesis was the translation of agent-state diagrams and various agent communications into the FLAME GPU framework. The use of FLAME GPU for the implementation of the basic immune simulator utilized the computational power of the GPUs via optimized CUDA code to achieve significant improvement in computational performance compared to a previous serial implementation using the RePast agent modeling toolkit. Statistical comparison between the parallel FLAME GPU implementation and the original repast implementation was done for immune win condition to validate statistical accuracy of the implementation. The results show that the parallel implementation using the FLAME GPU framework matched the results of the original RePast implementation within statistical limits. Therefore, it was shown that parallelization does not effect model accuracy.

In the current implementation of immune simulator, the diffusion of chemical signals was carried using 2D convolution stencil as described in the paper by Folcik et al. However, it is recommended to solve actual diffusion, reaction and advection equation with PDEs in conjunction with Agent Based Models (ABMs). Furthermore, for more accurate representation of the immune agent interaction, the ad-hoc rules, currently present in the finite state models, can be replaced with rules based on actual chemical kinetics.

The current implementation could be further developed to represent the immune system at a much higher level of detail. The framework could then be modularized to enable simulation of various immune system related conditions. The goal of this exercise would be to enable easy simulation by mixing and matching various modules with virtually no coding to simulate a plethora of disease conditions. This would enable basic science researchers who have typically limited training in programming, let alone parallel programming to quickly test out various hypothesis with computer models with a sufficient level of granularity.

Bibliography

[1] P. Matzinger, "The danger model: a renewed sense of self.," *Science (New York, N.Y.)*, vol. 296, pp. 301–305, 2002.

[2] B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, and P. Walter, *Molecular Biology of the Cell*. 2002.

[3] Z. Pancer and M. D. Cooper, "The evolution of adaptive immunity.," *Annual review of immunology*, vol. 24, pp. 497–518, 2006.

[4] D. Noble, "The rise of computational biology.," *Nature reviews. Molecular cell biology*, vol. 3, pp. 459–463, 2002.

[5] H. Van Dyke Parunak, R. Savit, and R. L. Riolo, "Agent-based modeling vs. equation-based modeling: A case study and users' guide," in *Multi-Agent Systems and Agent-Based Simulation*, pp. 10–25, 1998.

[6] "The basic immune simulator." `http://digitalunion.osu.edu/r2/summer06/sass/`. Online; accessed 12/23/2014.

[7] V. A. Folcik, G. C. An, and C. G. Orosz, "The basic immune simulator: an agent-based model to study the interactions between innate and adaptive immunity.," *Theoretical biology & medical modelling*, vol. 4, p. 39, 2007.

[8] "The repast suite." `http://repast.sourceforge.net/`. Online; accessed 12/23/2014.

[9] M. J. North, N. T. Collier, and J. R. Vos, "Experiences creating three implementations of the repast agent modeling toolkit," 2006.

[10] "Flexible large-scale agent modelling environment." `http://www.flamegpu.com/index.php`. Online; accessed 12/23/2014.

[11] N. Boccara, *Modeling Complex Systems.* Springer Science & Business Media, 2010.

[12] D. M. Gordon, *Ants at Work: How an Insect Society is Organized.* W. W. Norton, 2000.

[13] A. Kay, "Applying optimal foraging theory to assess nutrient availability ratios for ants," *Ecology*, vol. 83, pp. 1935–1944, 2002.

[14] M. A. Jansen, "Introduction to agent based modeling."

[15] D. M. Gordon, "Dynamics of task switching in harvester ants," 1989.

[16] D. M. Gordon, "The organization of work in social insect colonies," 1996.

[17] S. Boinski and P. A. Garber, *On the Move: How and Why Animals Travel in Groups.* University of Chicago Press, 2000.

[18] J. K. Parrish and W. M. Hamner, *Animal Groups in Three Dimensions: How Species Aggregate.* Cambridge University Press, 1997.

[19] J. Toner and Y. Tu, "Long-range order in a two-dimensional dynamical xy model: How birds fly together.," *Physical review letters*, vol. 75, pp. 4326–4329, 12 1995.

[20] T. Vicsek, A. Czirók, E. Ben-Jacob, I. Cohen, and O. Shochet, "Novel type of phase transition in a system of self-driven particles.," *Physical review letters*, vol. 75, pp. 1226–1229, 8 1995.

[21] D. B. BAHR and M. BEKOFF, “Predicting flock vigilance from simple passerine interactions: modelling with cellular automata,” *Animal Behaviour*, vol. 58, pp. 831–839, 10 1999.

[22] V. Grimm and S. F. Railsback, *Individual-based Modeling and Ecology.* Princeton University Press, 2005.

[23] C. M. Macal and M. J. North, “Agent-based modeling and simulation,” in *Proceedings of the 2009 Winter Simulation Conference (WSC)*, pp. 86–98, IEEE, 12 2009.

[24] E. Bonabeau, “Agent-based modeling: methods and techniques for simulating human systems.,” *Proceedings of the National Academy of Sciences of the United States of America*, vol. 99 Suppl 3, pp. 7280–7, 5 2002.

[25] J. Casti, *Would-Be Worlds: How Simulation is Changing the Frontiers of Science.* Wiley, 1997.

[26] N. R. Jennings, “On agent-based software engineering,” *Artificial Intelligence*, vol. 117, pp. 277–296, 3 2000.

[27] S. F. Railsback and V. Grimm, *Agent-Based and Individual-Based Modeling: A Practical Introduction*, vol. 6. Princeton University Press, 2011.

[28] S. Wolfram, “Statistical mechanics of cellular automata,” *Reviews of Modern Physics*, vol. 55, pp. 601–644, 7 1983.

[29] P.-O. Siebers and U. Aickelin, *Encyclopedia of Decision Making and Decision Support Technologies.* IGI Global, 2008.

[30] M. Gardner, “Mathematical games: The fantastic combinations of john conway’s new solitaire game "life",” *Scientific American*, vol. 223, pp. 120–123, 1970.

[31] R. M. Axelrod, *The Evolution of Cooperation.* Basic Books, 2006.

[32] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," 1987.

[33] J. M. Epstein and R. Axtell, *Growing Artificial Societies: Social Science from the Bottom Up.* Brookings Institution Press, 1996.

[34] T. A. Kohler, G. J. Gumerman, and R. G. Reynolds, "Simulating ancient societies.," *Scientific American*, vol. 293, pp. 76–82, 7 2005.

[35] J. Tang, K. F. Ley, and C. A. Hunt, "Dynamics of in silico leukocyte rolling, activation, and adhesion.," *BMC systems biology*, vol. 1, p. 14, 1 2007.

[36] J. Tang and C. A. Hunt, "Identifying the rules of engagement enabling leukocyte rolling, activation, and adhesion," *PLoS Computational Biology*, vol. 6, p. e1000681, 2 2010.

[37] P. Caplat, M. Anand, and C. Bauch, "Symmetric competition causes population oscillations in an individual-based model of forest dynamics," *Ecological Modelling*, vol. 211, pp. 491–500, 3 2008.

[38] *Nature-Inspired Informatics for Intelligent Applications and Knowledge Discovery: Implications in Business, Science, and Engineering: Implications in Business, Science, and Engineering.* IGI Global, 2009.

[39] M. Niazi, A. Hussain, A. R. Baig, and S. Bhatti, "Simulation of the research process," in *2008 Winter Simulation Conference*, pp. 1326–1334, IEEE, 12 2008.

[40] H. P. N. Hughes, C. W. Clegg, M. A. Robinson, and R. M. Crowder, "Agent-based modelling and simulation: The potential contribution to organizational psychology," *Journal of Occupational and Organizational Psychology*, vol. 85, pp. 487–502, 9 2012.

[41] "Modeling platforms." https://www.openabm.org/page/modeling-platforms. Online; accessed 04/05/2015.

[42] "Ecolab." http://ecolab.sourceforge.net/. Online; accessed 04/05/2015.

[43] "Laboratory for simulation development." http://www.labsimdev.org/Joomla_1-3/. Online; accessed 04/05/2015.

[44] "Mason." http://cs.gmu.edu/~eclab/projects/mason/. Online; accessed 04/05/2015.

[45] "Netlogo." http://ccl.northwestern.edu/netlogo/docs/. Online; accessed 04/05/2015.

[46] "Pandora: An hpc agent based modelling framework." http://www.bsc.es/computer-applications/pandora-hpc-agent-based-modelling-framework. Online; accessed 04/05/2015.

[47] "Starlogo on the web." http://education.mit.edu/starlogo/. Online; accessed 04/05/2015.

[48] E. B. Barker, W. C. Barker, W. E. Burr, W. T. Polk, and M. E. Smid, "Recommendation for key management," 3 2007.

[49] "Cuda c programming guide." https://http://docs.nvidia.com/cuda/cuda-c-programming-guide/. Online; accessed 04/05/2015.

[50] S. Orts-Escolano, J. Garcia-Rodriguez, V. Morell, M. Cazorla, J. Azorin, and J. Garcia-Chamizo, "Parallel computational intelligence-based multi-camera surveillance system," *Journal of Sensor and Actuator Networks*, vol. 3, pp. 95–112, 4 2014.

[51] J. Ghorpade, "Gpgpu processing in cuda architecture," *Advanced Computing: An International Journal*, vol. 3, pp. 105–120, 1 2012.

[52] T. Williams, "Parallel processing platform opens bridge to high performance embedded systems," 2014.

[53] S. Eilenberg, "Automata, languages, and machines," 3 1974.

[54] M. Holcombe, "X-machines as a basis for dynamic system specification," *Software Engineering Journal*, vol. 3, p. 69, 3 1988.

[55] P. Richmond, "Flame gpu technical report and user guide," tech. rep.

[56] 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, pp. 334–47, 5 2010.

[57] A. Levitin, *Introduction to the Design and Analysis of Algorithms.* Addison Wesley, second ed., 2007.

[58] S. Green, "Cuda particles," *NVIDIA SDK Whitepaper*, 2007.

[59] P. Grasso, S. Gangolli, and I. Gaunt, *Essentials of Pathology for Toxicologists.* CRC Press, 2002.

[60] B. Alberts, A. Johnson, and J. Lewis, "The adaptive immune system," in *Molecular Biology of the Cell*, ch. 24, Garland Science, 4th ed., 2002.

[61] C. J. Janeway, P. Travers, and M. Walport, "The complement system and innate immunity," in *Immunobiology:The Immune System in Health and Disease*, Garland Science, 2001.

[62] P. Fisher, "The innate and adaptive immune systems."

[63] R. J. Boyton and P. J. Openshaw, "Pulmonary defences to acute respiratory infection," 2002.

[64] B. Gomperts, I. Kramer, and P. Tatham, *Signal Transduction*, vol. 18. 2002.

[65] M. K. Liszewski, T. C. Farries, D. M. Lublin, I. A. Rooney, and J. P. Atkinson, "Control of the complement system.," *Advances in immunology*, vol. 61, pp. 201–283, 1996.

[66] H. Rus, C. Cudrici, and F. Niculescu, "The role of the complement system in innate immunity.," *Immunologic research*, vol. 33, pp. 103–112, 2005.

[67] G. Mayer, "Innate (non specific) immunity," in *Microbiology and Immunology On-line Textbook*, ch. 1, 2006.

[68] C. B. Thompson, "Apoptosis in the pathogenesis and treatment of disease," *Science*, vol. 267, pp. 1456–1462, 1995.

[69] A. Hoffbrand, J. Pettit, and P. Moss, *Essential Haematologu.* Blackwell Science, fourth ed., 2005.

[70] C. Little, H. Fowler, and J. Coulson, *The Shorter Oxford English Dictionary.* Oxford University Press (Guild Publishing), 1983.

[71] J. Ernst and O. Stendahl, *Phagocytosis of Bacteria and Bacterial Pathogenicity.* Cambridge University Press, 2006.

[72] K. Todar, "The phagocytic response of the host."

[73] J. A. Langermans, W. L. Hazenbos, and R. van Furth, "Antimicrobial functions of mononuclear phagocytes.," *Journal of immunological methods*, vol. 174, pp. 185–194, 1994.

[74] A. Ryter, "Relationship between ultrastructure and specific functions of macrophages.," *Comparative immunology, microbiology and infectious diseases*, vol. 8, pp. 119–133, 1985.

[75] R. B. Birge and D. S. Ucker, "Innate apoptotic immunity: the calming touch of death.," *Cell death and differentiation*, vol. 15, pp. 1096–1102, 2008.

[76] F. Krombach, S. Münzing, A. M. Allmeling, J. T. Gerlach, J. Behr, and M. Dörger, "Cell size of alveolar macrophages: an interspecies comparison.," *Environmental health perspectives*, vol. 105 Suppl, pp. 1261–1263, 1997.

[77] D. A. Ovchinnikov, "Macrophages in the embryo and beyond: Much more than just giant phagocytes," 2008.

[78] C. D. Mills, "M1 and m2 macrophages: Oracles of health and disease.," *Critical reviews in immunology*, vol. 32, pp. 463–88, 2012.

[79] V. Stvrtinova, J. Jakubovsky, and I. Hulin, "Neutrophils, central cells in acute inflammation," in *Inflammation and Fever from Pathophysiology:Principles of Disease*, Slovak Academey of Sciences:Academic Electronic Press, 1995.

[80] P. Delves, S. Martin, D. Burton, and I. Roit, *Roitt's Essential Immunology*. Blackwell Publishing, 2006.

[81] L. Sompayrac, *How the Immune System Works*. Blackwell Publishing, third ed., 2008.

[82] R. M. Steinman and Z. A. Cohn, "Identification of a novel cell type in peripheral lymphoid organs of mice. i. morphology, quantitation, tissue distribution.," *J Exp Med*, vol. 137, pp. 1142–62, 1973.

[83] P. Guermonprez, J. Valladeau, L. Zitvogel, C. Théry, and S. Amigorena, "Antigen presentation and t cell stimulation by dendritic cells.," *Annual review of immunology*, vol. 20, pp. 621–667, 2002.

[84] F. Sallusto and A. Lanzavecchia, "The instructive role of dendritic cells on t-cell responses.," *Arthritis research*, vol. 4 Suppl 3, pp. S127–S132, 2002.

[85] R. Steinman, "Dendritic cells," 2014.

[86] G. Krishnaswamy, O. Ajitawi, and D. S. Chi, “The human mast cell: an overview.,” *Methods in molecular biology (Clifton, N.J.)*, vol. 315, pp. 13–34, 2006.

[87] D. Price, “What is immunology?.”

[88] A. Iannello, O. Debbeche, S. Samarani, and A. Ahmad, “Antiviral nk cell responses in hiv infection: I. nk cell receptor genes as determinants of hiv resistance and progression to aids.,” *Journal of leukocyte biology*, vol. 84, pp. 1–26, 2008.

[89] E. Vivier, D. H. Raulet, A. Moretta, M. A. Caligiuri, L. Zitvogel, L. L. Lanier, W. M. Yokoyama, and S. Ugolini, “Innate or adaptive immunity? the example of natural killer cells.,” *Science (New York, N.Y.)*, vol. 331, pp. 44–49, 2011.

[90] *Acquired Immunity.* The American Heritage Medical Dictionary, 2007.

[91] C. Janeway, P. Travers, M. Walport, and M. Shlomchik, *Immunobiology.* Garland Science, fifth ed., 2001.

[92] T. J. Kindt, R. A. Goldsby, and B. A. Osborne, *Kuby Immunology*, vol. 6. 2007.

[93] “Understanding the immune system,” tech. rep.

[94] C. Janeway, P. Travers, M. Walport, and M. Shlomchik, *Immunobiology.* Garland Science, sixth ed., 2005.

[95] K. Shortman and S. H. Naik, “Steady-state and inflammatory dendritic-cell development.,” *Nature reviews. Immunology*, vol. 7, pp. 19–30, 2007.

[96] A. Beilhack and S. G. Rockson, “Immune traffic: a functional overview.,” *Lymphatic research and biology*, vol. 1, pp. 219–234, 2003.

[97] P. Srivastava, “Roles of heat-shock proteins in innate and adaptive immunity.,” *Nature reviews. Immunology*, vol. 2, pp. 185–194, 2002.

[98] Y. Shi, J. E. Evans, and K. L. Rock, “Molecular identification of a danger signal that alerts the immune system to dying cells.,” *Nature*, vol. 425, pp. 516–521, 2003.

[99] V. Wittamer, J.-D. Franssen, M. Vulcano, J.-F. Mirjolet, E. Le Poul, I. Migeotte, S. Brézillon, R. Tyldesley, C. Blanpain, M. Detheux, A. Mantovani, S. Sozzani, G. Vassart, M. Parmentier, and D. Communi, “Specific recruitment of antigen-presenting cells by chemerin, a novel processed ligand from human inflammatory fluids.,” *The Journal of experimental medicine*, vol. 198, pp. 977–985, 2003.

[100] R.-F. Guo and P. A. Ward, “Role of c5a in inflammatory responses.,” *Annual review of immunology*, vol. 23, pp. 821–852, 2005.

[101] A. Casadevall and L.-a. Pirofski, “Antibody-mediated regulation of cellular immunity and the inflammatory response.,” *Trends in immunology*, vol. 24, pp. 474–8, 9 2003.

[102] G. Ricevuti, “Host tissue damage by phagocytes.,” *Annals of the New York Academy of Sciences*, vol. 832, pp. 426–48, 12 1997.

[103] A. W. Segal, “How neutrophils kill microbes.,” *Annual review of immunology*, vol. 23, pp. 197–223, 1 2005.

[104] M.-L. N. Huynh, V. A. Fadok, and P. M. Henson, “Phosphatidylserine-dependent ingestion of apoptotic cells promotes tgf-beta1 secretion and the resolution of inflammation.,” *The Journal of clinical investigation*, vol. 109, pp. 41–50, 1 2002.

[105] S. Gallucci, M. Lolkema, and P. Matzinger, "Natural adjuvants: endogenous activators of dendritic cells.," *Nature medicine*, vol. 5, pp. 1249–55, 11 1999.

[106] F. Koch, U. Stanzl, P. Jennewein, K. Janke, C. Heufler, E. Kämpgen, N. Romani, and G. Schuler, "High level il-12 production by murine dendritic cells: upregulation via mhc class ii and cd40 molecules and downregulation by il-4 and il-10.," *The Journal of experimental medicine*, vol. 184, pp. 741–746, 1996.

[107] E. I. Zuniga, D. B. McGavern, J. L. Pruneda-Paz, C. Teng, and M. B. A. Oldstone, "Bone marrow plasmacytoid dendritic cells can differentiate into myeloid dendritic cells upon virus infection.," *Nature immunology*, vol. 5, pp. 1227–1234, 2004.

[108] C. F. Anderson, M. Lucas, L. Gutiérrez-Kobeh, A. E. Field, and D. M. Mosser, "T cell biasing by activated dendritic cells.," *Journal of immunology (Baltimore, Md. : 1950)*, vol. 173, pp. 955–961, 2004.

[109] P. L. Vieira, E. C. de Jong, E. A. Wierenga, M. L. Kapsenberg, and P. KaliÅĎski, "Development of th1-inducing capacity in myeloid dendritic cells requires environmental instruction.," *Journal of immunology (Baltimore, Md. : 1950)*, vol. 164, pp. 4507–4512, 2000.

[110] D. N. Hart, "Dendritic cells: unique leukocyte populations which control the primary immune response.," *Blood*, vol. 90, pp. 3245–3287, 1997.

[111] W.-S. Hou and L. Van Parijs, "A bcl-2-dependent molecular timer regulates the lifespan and immunogenicity of dendritic cells.," *Nature immunology*, vol. 5, pp. 583–589, 2004.

[112] A. J. Miga, S. R. Masters, B. G. Durell, M. Gonzalez, M. K. Jenkins, C. Maliszewski, H. Kikutani, W. F. Wade, and R. J. Noelle, "Dendritic cell longevity

and t cell persistence is controlled by cd154-cd40 interactions," *European Journal of Immunology*, vol. 31, pp. 959–965, 2001.

[113] E. Ingulli, A. Mondino, A. Khoruts, and M. K. Jenkins, "In vivo detection of dendritic cell antigen presentation to cd4(+) t cells.," *The Journal of experimental medicine*, vol. 185, pp. 2133–2141, 1997.

[114] E. Kriehuber, W. Bauer, A. S. Charbonnier, D. Winter, S. Amatschek, D. Tamandl, N. Schweifer, G. Stingl, and D. Maurer, "Balance between nf-??b and jnk/ap-1 activity controls dendritic cell life and death," *Blood*, vol. 106, pp. 175–183, 2005.

[115] D. B. Stetson, M. Mohrs, R. L. Reinhardt, J. L. Baron, Z.-E. Wang, L. Gapin, M. Kronenberg, and R. M. Locksley, "Constitutive cytokine mrnas mark natural killer (nk) and nk t cells poised for rapid effector function.," *The Journal of experimental medicine*, vol. 198, pp. 1069–1076, 2003.

[116] D. Amsen, J. M. Blander, G. R. Lee, K. Tanigaki, T. Honjo, and R. A. Flavell, "Instruction of distinct cd4 t helper cell fates by different notch ligands on antigen-presenting cells," *Cell*, vol. 117, pp. 515–526, 2004.

[117] H. Tanaka, C. E. Demeure, M. Rubio, G. Delespesse, and M. Sarfati, "Human monocyte-derived dendritic cells induce naive t cell differentiation into t helper cell type 2 (th2) or th1/th2 effectors. role of stimulator/responder ratio.," *The Journal of experimental medicine*, vol. 192, pp. 405–412, 2000.

[118] C. F. Anderson and D. M. Mosser, "Cutting edge: biasing immune responses by directing antigen to macrophage fc gamma receptors.," *Journal of immunology (Baltimore, Md. : 1950)*, vol. 168, pp. 3697–3701, 2002.

[119] D. R. Green, N. Droin, and M. Pinkoski, "Activation-induced cell death in t cells.," *Immunological reviews*, vol. 193, pp. 70–81, 2003.

[120] V. P. Badovinac, K. A. N. Messingham, A. Jabbari, J. S. Haring, and J. T. Harty, “Accelerated cd8+ t-cell memory and prime-boost response after dendritic-cell vaccination.,” *Nature medicine*, vol. 11, pp. 748–756, 2005.

[121] B. Dubois, B. Vanbervliet, J. Fayette, C. Massacrier, C. Van Kooten, F. Briere, J. Banchereau, and C. Caux, “Dendritic cells enhance growth and differentiation of cd40-activated b lymphocytes,” *J Exp Med*, vol. 185, pp. 941–951, 1997.