FÁBIO RODRIGUES MARTINS

SIMULAÇÃO DO SISTEMA IMUNOLÓGICO HUMANO POR MEIO DE MODELAGEM MULTIAGENTE PARALELA

Dissertação apresentada à Universidade Federal de Viçosa, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, para obtenção do título de *Magister Scientiae.*

VIÇOSA
MINAS GERAIS – BRASIL
2015

Ficha catalográfica preparada pela Biblioteca Central da Universidade Federal de Viçosa - Câmpus Viçosa

T
M386s
2015

Martins, Fábio Rodrigues, 1977-
Simulação do sistema imunológico humano por meio de modelagem multiagente paralela / Fábio Rodrigues Martins. – Viçosa, MG, 2015.
xiv, 85f. : il. (algumas color.) ; 29 cm.

Inclui apêndices.
Orientador: Alcione de Paiva Oliveira.
Dissertação (mestrado) - Universidade Federal de Viçosa.
Referências bibliográficas: f.77-79.

1. Programação paralela (Computação). 2. Inteligência artificial. 3. Agentes inteligentes (Software). 4. Sistema imunológico - Métodos de simulação. I. Universidade Federal de Viçosa. Departamento de Informática. Programa de Pós-graduação em Ciência da Computação. II. Título.

CDD 22. ed. 006.338

FÁBIO RODRIGUES MARTINS

SIMULAÇÃO DO SISTEMA IMUNOLÓGICO HUMANO POR MEIO DE MODELAGEM MULTIAGENTE PARALELA

Dissertação apresentada à Universidade Federal de Viçosa, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, para obtenção do título de *Magister Scientiae.*

APROVADA: 04 de dezembro de 2015.

Ricardo dos Santos Ferreira
Coorientador

Rodrigo Siqueira Batista

Alcione de Paiva Oliveira
Orientador

Dedico esta dissertação aos meus pais,
Lindaura e Francisco;
A minha esposa
Ezilane;
A minha filha
Joana.

"Vida real é viver para o outro."
(Bruce Lee)

Agradecimentos

Primeiramente, agradeço a Deus por me dar forças para superar os desafios que surgiram durante este período. Não foram poucos.

Agradeço a minha mãe Lindaura Rodrigues Martins e ao meu pai Francisco Alves Martins, por todo apoio.

A minha esposa Ezilane Ramalho Martins, pelo apoio, pelas mensagens positivas, e principalmente, por confiar que tudo iria dar certo.

A minha filha de apenas 2 anos que não tem consciência do quanto contribuiu para que este trabalho chegasse até o fim (desculpa pela ausência do papai!).

Ao meu orientador Alcione de Paiva Oliveira, por acreditar neste trabalho. O Sr. é um exemplo de profissional e de pessoa. Foi uma honra trabalhar a seu lado, agradeço por tudo. Muito Obrigado.

Aos professores Fábio Ribeiro Cerqueira, Ricardo dos Santos Ferreira, Levi Henrique Santana de Lelis e demais professores do corpo docente do Departamento de Informática da Universidade Federal de Viçosa.

Agradeço a todos os funcionário do DPI, ao pessoal da limpeza, da manutenção dos micros e etc. Em especial, ao Altino Alves de Sousa Filho, pela ajuda na parte burocrática do mestrado.

Agradeço ao Edson Rodrigues da Fonseca da "Processa Informática Juiz de Fora", pelo tempo que trabalhei nesta empresa, muitos dos conhecimentos aproveitei no mestrado. Aos colegas e amigos de trabalho, pelo incentivo no início do mestrado.

A todos os colegas do DPI, obrigado pelas conversas e oportunidade de descontração.

À Universidade Federal de Viçosa.

À CAPES pelo apoio financeiro.

Enfim, agradeço a todos que direta ou indiretamente, contribuíram para esta dissertação. Do fundo do meu coração obrigado!

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

ABMS - *Agent-based modeling and simulation*
APC - *Antigen-presenting cell*
API - *Application Programming Interface*
BIS - *The Basic Immune Simulator*
CD - *Cluster of diferenciation*
GPGPU - *General Purpose Graphic Processing Units*
GPU - *Graphics Processing Unit*
HPC - *High Performance Computing*
IA - Inteligência Artificial
IAD - Inteligência Artificial Distribuida.
MHC-I - Complexo de histocompatibilidade principal da classe I
MHC-II - Complexo de histocompatibilidade principal da classe II
NK - *Natural Killer*
PAMP - *Pathogen-associated Mecular Patterns*
PRR - *Pattern Recognition Receptors*
SI - Sistema imunológico
SIMD - *Single Struction Multiple Data*
SM - *Stream Multiprocessors*
SMA - Sistema multiagente
XML - *Xtensible Markup Language*

Resumo

MARTINS, Fábio Rodrigues, M.Sc., Universidade Federal de Viçosa, dezembro de 2015. **Simulação do sistema imunológico humano por meio de modelagem multiagente paralela** Orientador: Alcione de Paiva Oliveira. Coorientadores: Fábio Ribeiro Cerqueira e Ricardo dos Santos Ferreira.

Este trabalho apresenta uma proposta de modelagem do sistema imunológico (SI) humano. Mais especificamente, propõe a modelagem do SI por meio de sistema multiagente paralelo. O ser humano é exposto a uma quantidade imensurável de agentes não-próprios no decorrer de um dia. O fato de não sermos afetados pelos mesmos se deve ao SI, uma vez que atua para manter a homeostase (equilíbrio orgânico). Então, estudar o comportamento deste sistema é fundamental, já que descobertas nesta área impactam na vida de todas as pessoas. Uma forma de investigar o comportamento do SI é por meio de simulações computacionais (experimentação *in-silico*). Mas, como o SI é grande e complexo, demanda muito processamento. Esta característica impõe algumas restrições para estas simulações, já que até o momento uma geração de *frameworks* que estava disponível, no mercado, eram os ABMS (do inglês *Agent-basead mod-eling and simulation*), que são indicados para testes mais simples. Por este motivo, neste trabalho foi utilizado o *framework* Flame que se enquadra na geração HPC (do inglês *High Performance Computing*). Este *framework* é usado para programação paralela com alto poder computacional. No entanto, para agilizar ainda mais o resultado dos experimentos, em uma parte do modelo, foi utilizada a programação para placa gráfica (GPU). A comparação entre a implementação deste trabalho e de outro SI artificial - o *AutoSimmune* aponta que a abordagem multiagente paralelo é superior aos ABMS antigos.

Abstract

MARTINS, Fábio Rodrigues, M.Sc., Universidade Federal de Viçosa, December, 2015. **Simulation of the human immune system in modeling multi-agent means of parallel.** Adviser: Alcione de Paiva Oliveira. Co-advisor: Fábio Ribeiro Cerqueira and Ricardo dos Santos Ferreira.

The research presented this dissertation deals with the human immune system (IS) simulation. More specifically, about modeling the IS by parallel multi-agent systems. Human beings are exposed to an immeasurable number of threatening microorganisms everyday. The fact of not being affected by these same is due to the IS, since it operates to maintain homeostasis (organic balance). Thus the study this system behavior is essential, as discoveries in this area may have impact on the lives of all people. One way to investigate the IS behavior is by means of computer simulations (experiment in-silico). But as the IS is very large and complex it requires a lot of computing power.The emergence of agent oriented systems has provided an alternative approach to address many complex problems similar to the immune system, that requires distributed behavior, local decisions, and emerging global behavior from the interactions of their basic elements. However, despite providing a suitable tool for modeling complex distributed systems, implementations of multi-agent systems are limited by the available hardware architecture. A recent possibility to circumvent this problem is the use of graphics cards to implement such systems. Nevertheless, these devices reach the optimal performance when agents have homogeneous and simple behavior, which might not be the case of many problems. Systems such as simulators of the immune system, in addition to having a large number of agents with complex behavior, those agents communicate massively, indirectly, through dissemination of various substances in their environment. Diffusion of substances is something easily simulated in modern current graphics cards, but the problem is to provide the results of those simulations to thousands (or millions) of agents

simultaneously. Therefore in this study we used the Flame framework. This framework is used for parallel programming with self computational power. However, to further expedite the result of the experiment, in a part of the model program was used for the graphic card. The comparison between the implementation of this work and another immune system points out that the parallel multi-agent approach is superior to the sequential implementation.

Capítulo 1

Introdução

A modelagem e simulação baseada em agentes (ABMS - do inglês *Agent-based modeling and simulation*) tem sido amplamente utilizada por diversas áreas como: bioinformática, ciências sociais, entre outras, na modelagem de sistemas complexos. Neste contexto, até o momento, os *frameworks* que implementam esta técnica têm auxiliado os pesquisadores nos testes de suas hipóteses (experimentação *in-silico*). Mas, segundo Collier & North [2012], estes sistemas deverão ser reformulados, pois os problemas que estão surgindo demandam uma capacidade computacional cada vez maior. Portanto, os ABMS tradicionais que utilizam processamento essencialmente sequencial deverão ser utilizados apenas em simulação de problemas simples. Já em situações em que se exige um grande poder computacional, será necessário utilizar um novo conceito que é o *High Performance Computing* (HPC). Esta plataforma tem seu foco em computação distribuída de larga escala.

A verificação de acontecimentos, na realidade, traz mais veracidade para pesquisa; porém existem casos em que a observação destes é inviável, uma vez que não é possível reproduzir a situação no mundo real. Este tipo de problema é mais adequado para ser simulado computacionalmente, ou seja testado (*in-silico*).

Possi et al. [2011] apresentou um simulador do sistema imunológico (SI) e o chamou de *AutoSimmune* que foi desenvolvido baseado no *The Basic Immune Simulator* (BIS) (Folcik et al. [2007]). Estes sistemas além de ter em comum a sua aplicação que é exatamente a simulação do SI, também utilizam o paradigma de sistemas multiagentes reativo massivo. O *AutoSimmune* foi desenvolvido utilizando o *framework Repast Simphony* versão 1.2.0 que usa a linguagem de programação Java.

Segundo Sousa et al. [2014] a evolução do *AutoSimmune* vai acontecer naturalmente. Assim, à medida que novos conceitos vão sendo inseridos no simulador, a

complexidade do mesmo aumenta, juntamente com a quantidade de dados a serem processados. Então, para que os experimentos sejam realizados com mais pureza e assertividade, faz-se necessário utilizar técnicas de programação paralela nas simulações.

1.1 O problema e sua importância

Inúmeras doenças com envolvimento do sistema imunológico podem ser investigadas por meio de simulação computacional (*in-silico*), mas neste trabalho vamos nos ater às doenças autoimunes. Possi et al. [2011] desenvolveram um simulador baseado no trabalho de Folcik et al. [2007], o qual passou a ser denominado *AutoSimmune*. Este foi aplicado a diversos fenômenos biológicos relacionados com o sistema imune (SI) humano, tais como o desenvolvimento da autoimunidade.

Outros autores se basearam no modelo inicial do *AutoSimmune* e desenvolveram novos trabalhos como: (1) verificação do papel do mastócito em infecções, Silva et al. [2012]; (2) estudo da resposta imune na glomerulonefrite pós-infecciosa (GnPE) por *Streptococcus pyogenes*, Bastos et al. [2013]; (3) verificação do papel dos Linfócitos T CD4+CD25+ na regulação do sistema imunológico sob a perspectiva ocorrência da sepse, Siqueira-Batista et al. [2010]; e (4) simulação da resposta imune na sepse, [Sousa et al., 2014].

No entanto, apesar das simulações realizadas terem obtido resultados importantes, ficou claro que o simulador limitava algumas e impedia que outras com número muito grande de células fossem realizadas. Mesmo a utilização de máquinas modernas com mais recursos computacionais não contribui para o desempenho na execução dos experimentos. Este é um ponto crítico, uma vez que a demanda por simulações mais complexas vem crescendo. Simulações nas quais em uma única iteração (denominada comumentemente como *tick*) serão processados milhões de agentes representando células e elementos biológicos e químicos, com o intuito de verificar a interação entre eles e o comportamento do SI como um todo.

A evolução das pesquisas está relacionada com a quantidade e a qualidade dos experimentos. Então, se alguma limitação for imposta, ou se a simulação não for realizada seguindo os padrões da literatura, o resultado da pesquisa estará comprometido, impedindo assim novos ensaios mais complexos. Portanto, foi observado que o *AutoSimmune* já não estava atendendo aos quesitos de quantidade de agentes e tempos de resposta dos experimentos.

Para que avanços sejam possíveis é preciso desenvolver uma arquitetura de

hardware e de software que seja escalável em relação ao número de agentes, que se comunicam de forma intensiva. A combinação do poder de processamento das placas gráficas com *clusters* dotados de processadores de alto desempenho parece ser o caminho mais promissor.

1.2 Hipótese

O uso do paradigma multiagente em ambiente de programação paralela permite a simulação mais realista do sistema imunológico humano do que a abordagem sequencial. Esta afirmação poderá ser comprovada por meio de simulação paralela de algumas funções do SI, que quando comparadas com a equivalente sequencial se mostrarão mais realistas.

1.3 Objetivos

O objetivo geral do trabalho é propor um modelo de simulação do SI que utilize agentes inteligentes e funcione em uma arquitetura paralela. Especificamente pretende-se:

- desenvolver um novo simulador do sistema imune humano;
- comparar o simulador desenvolvido com o *AutoSimmune*.

1.4 Organização da dissertação

Esta dissertação possui, além da introdução, mais 4 capítulos. O primeiro Capítulo é esse que ora está sendo lido. No Capítulo 2, estão descritos os principais conceitos que nortearão o desenvolvimento do modelo, ou seja, o referencial teórico. Mais especificamente na sessão 2.1 são apresentados os conceitos gerais do SI, como imunidade inata, humoral, celular, adaptativa, adaptativa humoral, adaptativa celular e autoimunidade. Na seção 2.2 é apresentado o conceito de sistemas multiagente. Na seção 2.3, é apresentado o conceito de programação para GPU e a GPGPU. Na seção 2.4, são apresentados os aspectos funcionais de alguns *frameworks*. Enfim, a seção 2.5 apresenta os trabalhos correlatos.

O Capítulo 3 descreve o modelo proposto neste trabalho e seus requisitos, assim como o *framework* alvo. Também estão presentes neste capítulo, seções que descrevem as zonas, o procedimento de difusão de substâncias e os agentes.

No Capítulo 4, são apresentados os resultados dos experimentos, a avaliação destes resultados e a análise dos mesmos, onde são evidenciados os melhores e piores resultados.

Finalmente, no Capítulo 5 é apresentada a conclusão obtida nesta dissertação e perspectivas para trabalhos futuros.

Capítulo 2

Referencial Teórico

2.1 O sistema imunológico

A palavra imunidade é derivada do termo latino *immunitas*, que se referia à proteção legal que os senadores romanos tinham durante seus mandatos. Em analogia com as leis atuais do Brasil, isto seria a imunidade parlamentar. No entanto, este termo passou a ser utilizado para se referir à proteção contra doenças infecciosas e outros agravos. O sistema imunológico (SI), por sua vez, é constituído de células e moléculas responsáveis pela imunidade. Já a resposta imunológica é a ação coordenada e coletiva na resposta aos agentes não-próprios que adentram no corpo humano [Abbas et al., 2012].

O SI tem como principal função a resposta às possíveis infecções causadas por: vírus, bactérias, fungos, protozoários e helmintos. Este grupo - quando implicado em doenças - compõem genericamente os patógenos. O SI reconhece o que é próprio do corpo atuando assim o não próprio. Em caso de reconhecimento do não próprio são iniciadas várias ações, as quais são nomeadas como resposta imunológica. A resposta imunológica é categorizada em dois grupos: imunidade inata, também chamada de natural ou nativa e imunidade adaptativa que pode ser específica ou adquirida [Abbas et al., 2012]. Outra função do SI é impedir ou controlar o crescimento de células próprias que foram alteradas por infecções, vírus ou mutação, com é o caso das células tumorais.

Com o intuito de alcançar a imunidade, o SI realiza a resposta imunológica das seguintes maneiras (tradicionalmente consideradas): quando uma célula do SI elimina um patógeno, esta resposta é denominada celular. Mas quando a eliminação é mediada por alguma molécula liberada por componentes do SI, ocorre a resposta imunológica humoral [Abbas et al., 2012; Floreano & Mattiussi, 2008].

De acordo com Abbas et al. [2012], a resposta contra micro-organismos é realizada de duas maneiras: a imunidade inata atua na primeira linha de defesa do organismo e caracteriza-se por ser mais ágil; já a imunidade adaptativa apresenta uma resposta mais tardia, porém mais eficaz. A Figura 2.1 apresenta os tipos de respostas descritos acima, com seu elementos e faixa de tempo após a infecção.

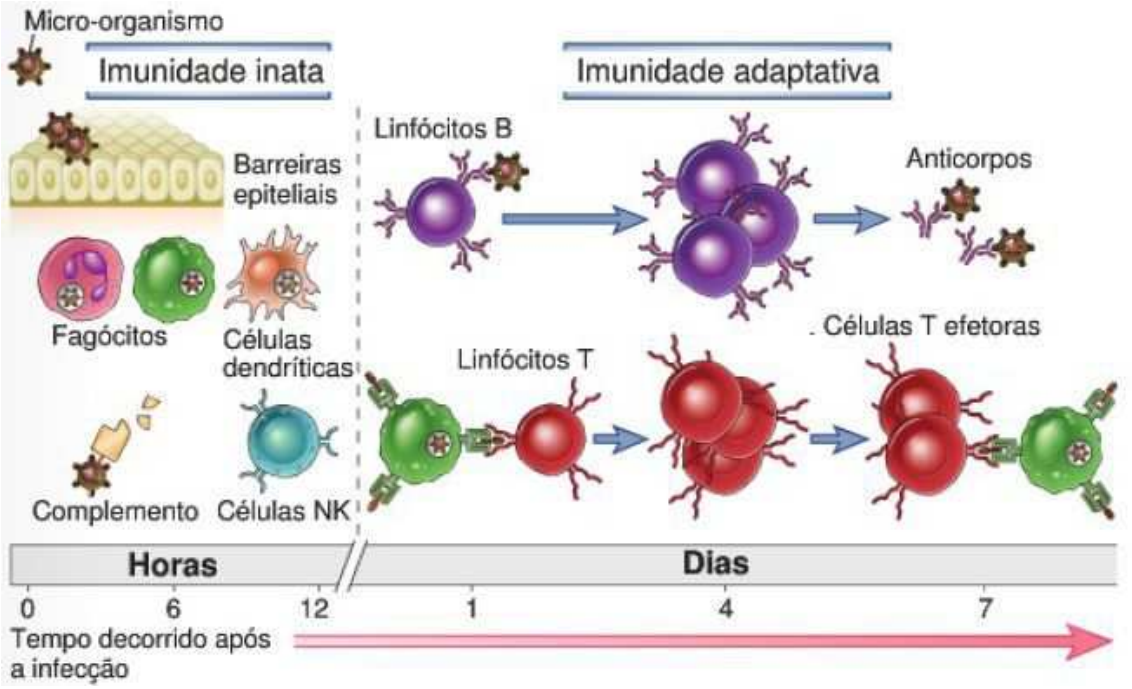

Figura 2.1. Imunidade inata e adaptativa.

Fonte: [Abbas et al., 2012]

2.1.1 Imunidade inata

Segundo os autores Abbas et al. [2012]; Peakman & Vergani [2011], a imunidade inata constitui a primeira ação de resposta do SI contra micro-organismos. Esta não guarda memória e não é específica. De modo que reage da mesma forma para todos os casos de infecção. Floreano & Mattiussi [2008] apresentam a imunidade inata sendo o conjunto de mecanismos de defesa que não se alteram durante a vida do organismo hospedeiro.

A etapa inicial da imunidade são as barreiras físicas e químicas que são compostas pela: pele (ácidos láticos e graxos); mucosas (muco impede a aderência do agente infeccioso à pele); as secreções que removem resíduos das superfícies mucosas; e os cílios que atuam na remoção de detritos e estruturas estranhas Peakman & Vergani [2011]. Estes mesmos autores declaram a existência de fatores imunológi-

cos humorais presentes na secreção da mucosa e no sangue. Ainda segundo este autor os principais componentes são: o complemento; a lectina ligante de manana; e as demais opsoninas (uma opsonima auxilia a digestão de bactérias por neutrófilos).

Uma vez que os agentes infecciosos superam a barreira física e invadem o agente hospedeiro, inicia-se a atividade das células fagocitárias (neutrófilos, mascrófagos), células dendríticas e células assassinas natural *natural killer* (NK). Estas células atuam na eliminação dos agentes infecciosos e no retardo da proliferação dos mesmos enquanto a imunidade adaptativa prepara sua resposta Peakman & Vergani [2011].

2.1.2 Imunidade inata humoral

Floreano & Mattiussi [2008]; Peakman & Vergani [2011] declaram que o componente mais importante da imunidade humoral é o sistema complemento. Este sistema é composto de um conjunto de moléculas presentes no organismo do hospedeiro. Estas moléculas têm afinidade com as superfícies das membranas celulares tanto dos hospedeiros quanto dos patógenos. Então, quando a reação se inicia, um grupo de moléculas se ligam à membrana plasmática, criando - assim - um orifício, o qual permite a entrada de substâncias do meio extracelular em grande quantidade na célula. A consequência é a lise celular.

Existem casos em que os agentes infecciosos evoluem e impedem que as moléculas do sistema complemento lise sua membrana. Mas esta evolução não impedirá que o mesmo seja fagocitado, pois o sistema complemento elimina qualquer elemento marcado pela imunidade inata celular [Floreano & Mattiussi, 2008; Siqueira-Batista et al., 2015].

2.1.3 Imunidade inata celular

Quando os patógenos adentram no organismo e começam a danificar células do tecido, há liberação de algumas substâncias, as quais iniciam o processo de recrutamento de leucócitos para responder ao processo infeccioso, o que ocorre no contexto da inflamação [Abbas et al., 2012]. Murphy [2010] aproveitando uma definição clássica, caracteriza a inflamação através dos sinais cardinais de calor, dor, rubor e edema que acontecem devido à liberação de citocinas e de outros mediadores inflamatórios nos vasos sanguíneos.

Delves et al. [2011] declaram a existência de dois tipos principais de células que atuam na resposta imunológica inata - os "fagócitos profissionais- os quais são: neutrófilos e macrófagos. Enquanto os neutrófilos atuam na fagocitose das bactérias

piogênicas e fungos, os macrófagos agem em bactérias, vírus e protozoários que infiltram e vivem no interior da célula hospedeira. Ambos se deslocam em direção a inflamação.Murphy [2010] também declara "Os neutrófilos e os macrófagos são as principais células inflamatórias".

Os neutrófilos são leucócitos polimorfonucleares e se caracterizam por terem maior população presente na corrente sanguínea em um ser humano adulto. São produzidas cerca de 1 x 10^{11} células por dia, as quais permanecem no sangue por seis horas. Sabe-se que eles são a primeira linha do SI no processo de resposta a uma infecção. Sua ação é ingerir patógenos e tem seu ciclo finalizado com a apoptose [Abbas et al., 2012].

Por outro lado, o macrófago é produzido em quantidade menor e realiza diversas funções, tanto na imunidade inata quanto na adquirida. Segundo Abbas et al. [2012], uma das funções principais dos macrófagos na resposta do hospedeiro é a ingestão e morte de micro-organismos. Esta prática libera substâncias que estimulam a cicatrização e outra que sinalizam para que o SI mobilize mais células de defesa. Além das funções citadas, os macrófagos também: fagocitam células mortas; atuam como células APC que apresentam antígenos para ativação dos linfócitos T e promovem o reparo do tecido lesionado.Vale lembrar que os antígenos substâncias identificadas como não-próprias, que ao serem reconhecidas pelo SI (inato e adaptativo), são capazes de desencadear resposta imune. Em alguns casos os constituintes do hospedeiro (por exemplo, humano) podem ser reconhecidos como não-próprios: estes são autoantígenos, implicados nos processos de autoimunidade.

Para que a resposta imunológica seja eficiente e não se volte para as células do próprio organismo, é necessário que exista uma forma de diferenciar o que é próprio do não-próprio, isto é, as células inofensivas das que são danosas ao ser humano. Esta diferenciação acontece devido aos receptores, uma vez que as células do SI possuem uma estrutura genética que se denominam receptores de reconhecimento de padrões, do inglês (*pattern recognition receptors* - PRR). Os PRR reconhecem os padrões moleculares relacionados aos patógenos (do inglês pathogen-associated mecular patterns - PAMPs), estruturas presentes nos micro-organismos. Deve ser destacado que um mesmo agente infeccioso pode gerar diversos epítopos com diversas afinidades, podendo ser reconhecido por diversos PRRs [Floreano & Mattiussi, 2008]. Qualquer agente infeccioso que seja identificado por um PAMP pode ser fagocitado por um macrófago ou neutrófilo.

Além dos agentes infecciosos, os tecidos do próprio organismo podem apresentar antígenos que são chamados de autoantígenos. Portanto, o SI reconhece os antígenos de patógenos e os autoantígenos e diferencia uns dos outros para certificar

o que é próprio e atacar o não próprio, ou seja, os patógenos Abbas et al. [2012].

Existem patógenos que burlam os processos citados acima e invadem as células hospedeiras onde os fagócitos não conseguem alcançá-los. Estes patógenos injetam seu código genético nestas células e as transformam em replicadoras. Este comportamento faz com que sejam criadas novas células semelhantes às infectadas e origina um depósito de infecção [Abbas et al., 2012]. Neste momento temos uma infecção por patógenos intracelular. Neste caso o SI dispõe de uma classe de leucócitos capaz de eliminar estas infecções. São as células exterminadoras naturais do inglês *Natural Killer* (NK). A célula NK identifica se um célula está infectada ou não. Em caso de infecção, são liberadas substâncias que ocasionam a morte programada da célula, evento conhecido como apoptose. A verificação da infecção celular é feita através do complexo de histocompatibilidade principal da classe I (MHC-I) que em inglês: (*class I major histocompatibility complex*). Todas as células nucleadas apresentam em sua superfície amostras das proteínas sintetizadas em seu interior. No caso das células não estarem infectadas serão apresentadas estas proteínas e MHC-I, que são proteínas próprias do organismo, neste caso a NK é inibida. Em outra situação onde as células apresentam proteínas sendo sintetizadas no seu interior diferente do organismo ou não apresentam MHC-I, entende-se que a célula está infectada. Sendo assim, a NK é ativada e lisa esta célula [Abbas et al., 2012]. Este comportamento é apresentado na Figura 2.2.

No entanto, além das funções mencionadas, os leucócitos também são responsáveis por outra atividade crucial que é a mobilização do SI adaptativo. Isto ocorre quando, os leucócitos não conseguem eliminar os patógenos e produzem uma grande quantidade de citocinas que desencadeiam uma resposta inflamatória que incita elementos da imunidade adaptativa [Floreano & Mattiussi, 2008], conforme será comentado a seguir.

2.1.4 Imunidade adaptativa

Como citado anteriormente, a imunidade adaptativa caracteriza-se por realizar a resposta imunológica tardia. Difere da imunidade inata, pois possibilita a geração de especificidade para antígenos distintos e memoriza (memória) as exposição anteriores aos patógenos. Nesta fase, entra em cena leucócitos especiais chamados linfócitos, que são classificadas como as principais células da imunidade adaptativa. Apesar destas células serem nomeadas como linfócitos, são diferenciadas de acordo com sua origem (timo, medula óssea) e as proteínas de suas superfícies, que recebem o nome de *cluster of differenciation* (CD). Os principais linfócitos são linfócitos B e

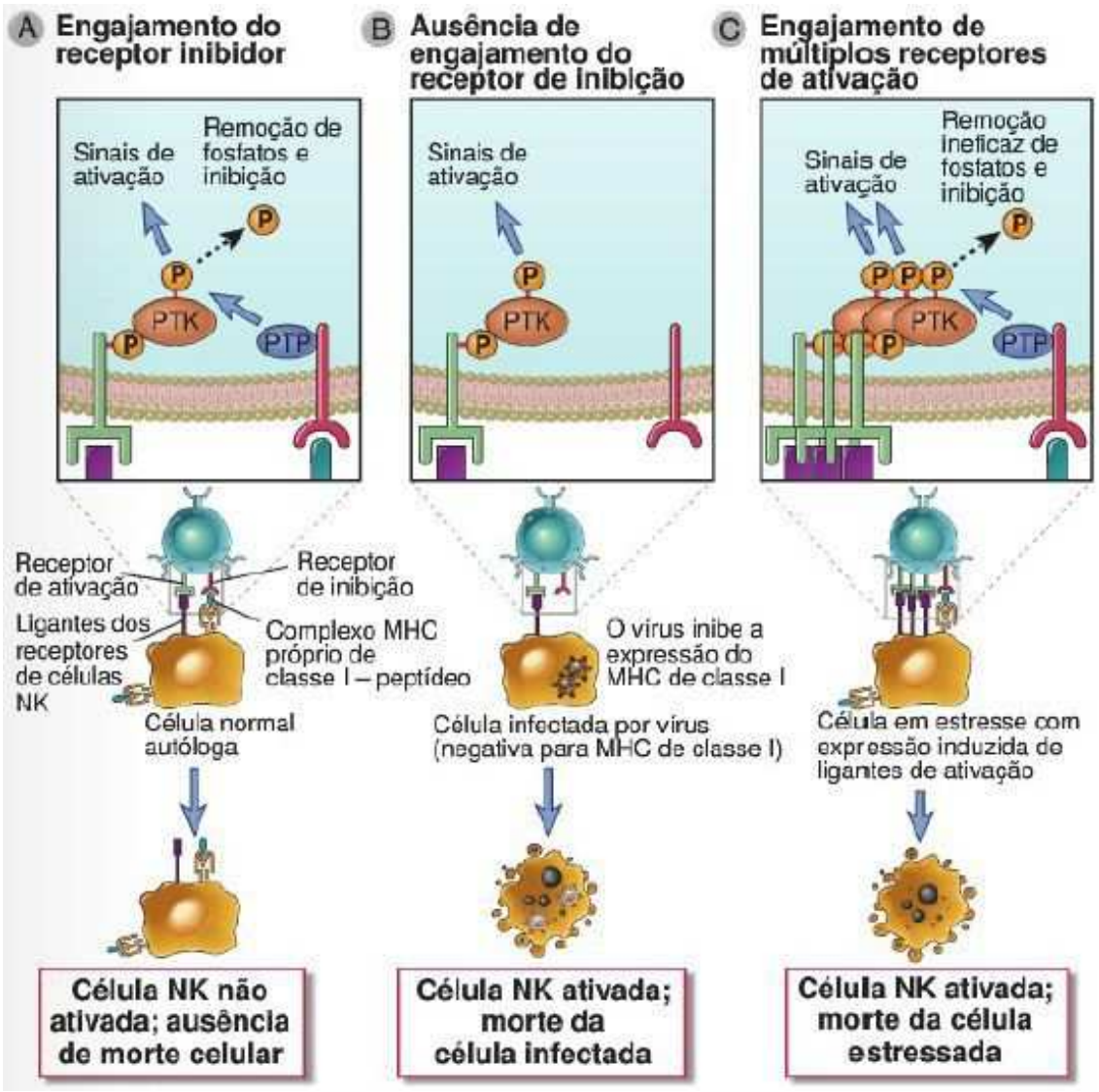

Figura 2.2. Funções dos receptores de ativação e inibição das células NK.

Fonte: [Abbas et al., 2012]

o linfócito T [Abbas et al., 2012].

O linfócito B é gerado de células tronco da medula óssea (*bone marrow*); por este motivo recebe a letra B no nome é o único linfócito gerador de anticorpos. Apto a reconhecer patógenos nocivos ao corpo e a acoplar ao mesmo, impedindo que se ligue a outras estruturas. Já o linfócito T é gerado no Timo. Possui diversos tipos, mas os que se destacam são o T CD4+ também chamados de linfócitos T auxiliares T *helper* (Th) que atuam ajudando os linfócitos B na produção de anticorpos, e as células fagocitárias na eliminação de patógenos.

Segundo Floreano & Mattiussi [2008], estes linfócitos possuem a capacidade de reconhecer qualquer antígeno, podendo gerar combinações de até 10^{18}. Porém, é

inviável gerar esta quantidade de linfócitos. Então o SI realiza a geração de linfócito com afinidade aleatória e ao passar do tempo renova a população. Abbas et al. [2012] define esta função como geração de diversidade.

Como os linfócitos são gerados aleatoriamente, eles não são colocados em circulação imediatamente, pois os mesmos podem ser auto-reativos e atacar antígenos do próprio organismo. Por isto, eles passam por uma etapa de seleção onde são identificados e eliminados os auto-reativos. O processo de eliminação é denominado tolerância central, apresentado na Figuras 2.3. Este processo é dividido em duas etapas sendo a primeira nomeada de seleção positiva. Consiste em verificar se o linfócito reconhece as moléculas MHC das células. Caso consiga, este é descartado; se não ele é aprovado no teste [Floreano & Mattiussi, 2008].

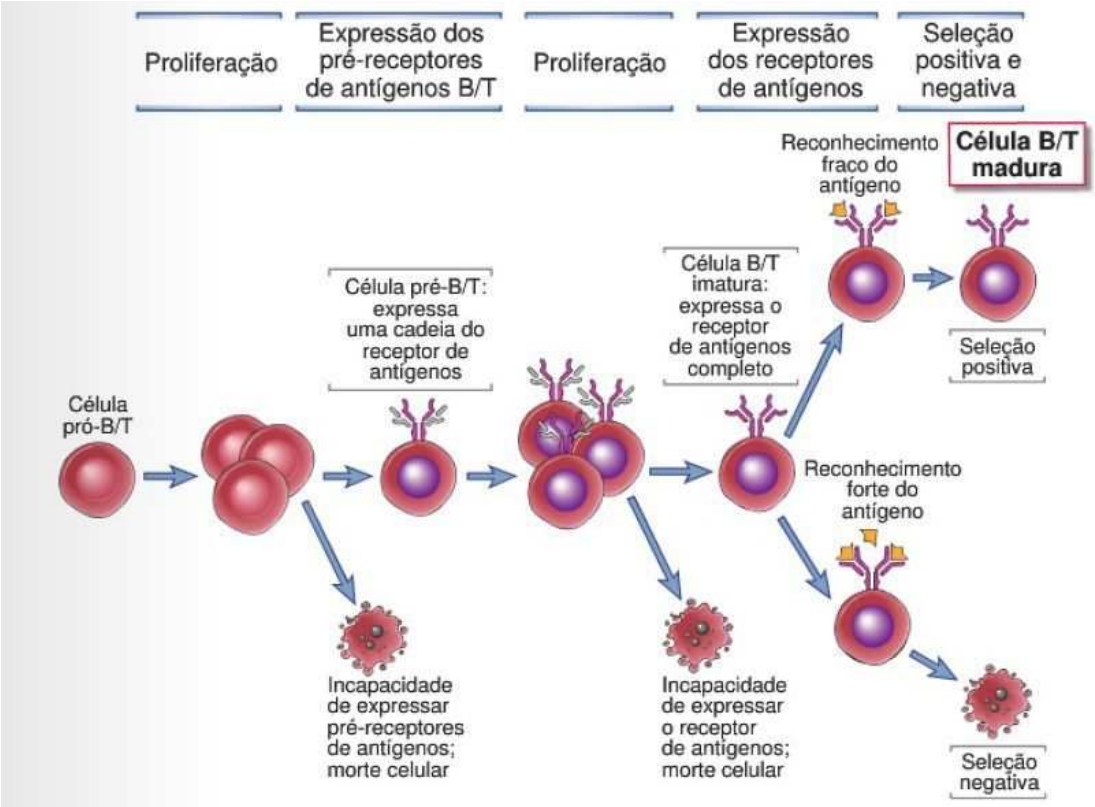

Figura 2.3. Ponto de controle na maturação dos linfócitos.

Fonte: [Abbas et al., 2012]

A outra etapa é a seleção negativa que consiste em mostrar o linfócito a uma molécula do MHC apresentando autoantígenos. Na hipótese de reconhecimento, fortemente o linfócito é eliminado. Mas quando o reconhecimento é fraco, o linfócito passa no teste, pois se considera que pode reconhecer outros antígenos mais fortes. Já o linfócito B é submetido apenas à etapa da seleção negativa [Floreano

& Mattiussi, 2008]. Ao finalizar as etapas anteriores, os linfócitos que passarem no teste são direcionados ao linfonodo, ou seja, para os órgãos linfoides periféricos (linfonodo, baço, as amígdalas entre outras), conforme a Figura 2.4.

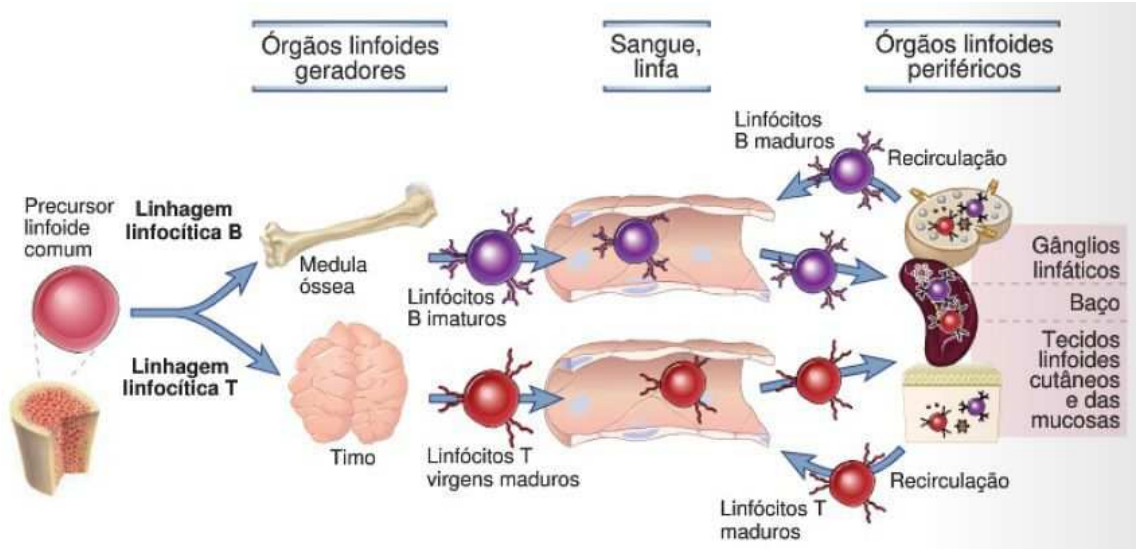

Figura 2.4. Maturação dos linfócitos.

Fonte: [Abbas et al., 2012]

Após serem submetidos à tolerância central e serem migrados para os órgãos linfoides, os linfócitos estão maduros, porém não ativos, devido ao fato que eles ainda podem ser auto-reativos a autoantígenos ausentes nos testes de tolerância, mas presentes em alguma parte do corpo humano [Floreano & Mattiussi, 2008]. A Figura **??** descreve o processo de ativação do linfócito, que seria a existência de dois sinais, o primeiro mediado pela detecção de antígenos (p. ex. microbianos).

Entretanto, encontrar um antígeno não é uma tarefa simples, e para auxiliar o linfócito nesta atividade o SI disponibiliza dois mecanismos, que possibilitem aglomerar os linfócitos e os antígenos nos órgãos linfoides periféricos. O primeiro mecanismo é a drenagem dos líquidos dos tecidos epiteliais, conjuntivo e parenquimatoso, que são denominados linfa, pelos vasos linfáticos até o linfonodo. Este líquido contém amostras de antígenos presentes nas imediações. Por outro lado, o baço atua de forma semelhante e filtra os patógenos presentes na circulação sanguínea [Abbas et al., 2012].

Já o outro mecanismo é: as células apresentadoras de antígenos "profissionais"*antigen-presenting cell* (APC) que capturam os antígenos em qualquer parte do organismo levando-os diretamente para o linfonodo, para serem apresentados aos linfócitos [Abbas et al., 2012]. As APCs possuem moléculas especiais

em sua superfície que são do complexo de histocompatibilidade principal da classe II (MHC-II). Além de capturar antígenos, estas células possuem a capacidade de verificar a presença do sinal 2 no local e informar aos linfócitos.

Em se tratando de células APC, as células dendríticas são consideradas as mais importantes e têm como função o "patrulhamento"do tecido em busca de antígenos. Possuem diversos receptores PRRs que lhe atribuem a capacidade de reconhecer vários tipos de *danger signals* (sinal de perigo). Ficam inativas até que encontrem um foco de sinais 2 e neste momento se ativam. Ao encontrarem antígenos, estas células migram para o linfonodo e apresentam aos linfócitos informações, dentre as quais se o antígeno foi capturado em uma *danger zone* (zona de perigo) [Abbas et al., 2012]. Apesar destas células estarem descritas nesta sessão, Abbas et al. [2012] declara que as mesmas tem papel importantíssimo na imunidade inata.

Outras células APC do SI são os macrófagos, que por fazerem parte deste conjunto, também possuem a habilidade de fagocitar patógenos e apresentar seus antígenos aos linfócitos. Na prática desta atividade, podem-se citar também os linfócitos B que são dotados da capacidade de apresentar antígenos aos linfócitos T [Abbas et al., 2012].

Quando os linfócitos são ativados, eles começam a se multiplicar, formando populações numerosas. O processo de crescimento é denominado expansão clonal. Segundo Abbas et al. [2012], o número de linfócitos T CTL de um antígeno específico presente em um organismo saudável é por volta de $1x10^6$. Porém, com a explosão clonal, este número cresce 20%.

A eliminação dos linfócitos acontece à medida em que os patógenos são fagocitados, pois o sinal 2 é reduzido impedindo o recrutamento de mais células. No entanto, o ciclo de vida dos linfócitos ativos, são finalizados caso fiquem algum tempo sem serem reativados ou atinjam o limite de tempo de sobrevivência. Segundo Abbas et al. [2012], o processo descrito acima (resposta imunológica) se completa em uma a duas semanas após a infecção. O SI retorna ao seu estado normal tendo como lembrança do ocorrido apenas os linfócitos de memória; desta forma, a memória imunológica do SI.

A memória imunológica permite que o SI dê uma resposta rápida, com mais intensidade e com mais precisão, pois já se sabe o padrão do antígeno. A cada encontro com antígenos, os linfócitos de memória são gerados. Esta técnica é utilizada para vacinas, nas quais é injetado um patógeno enfraquecido no corpo humano. As etapas citadas anteriormente são feitas e as células de memória são criadas. Quando ocorre um ataque pelo patógeno, este não será mais desconhecido [Abbas et al., 2012].

A Figura 2.5 apresenta os passos do SI em uma resposta imunológica com a criação das células de memória. Elas são: reconhecimento de antígenos, ativação dos linfócitos, eliminação dos antígenos, declínio, memória.

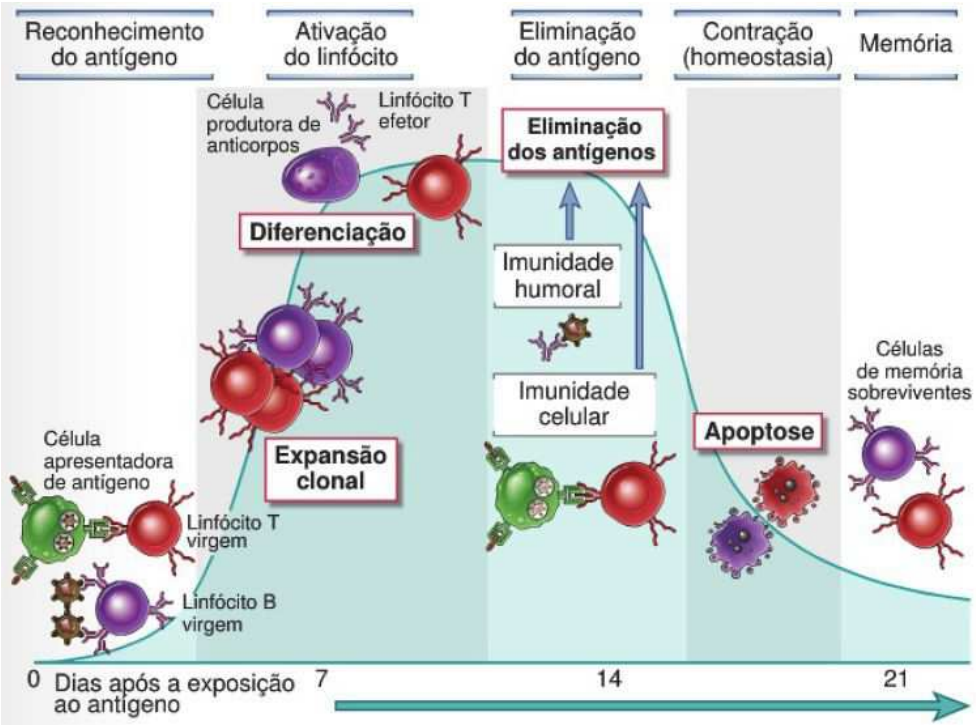

Figura 2.5. Fases das respostas imunológicas adaptativas.

Fonte: [Abbas et al., 2012]

Similar à imunidade inata, a imunidade adaptativa se ramifica em imunidade adaptativa celular e imunidade adaptativa humoral. A primeira ocorre quando o patógeno é fagocitado logo de início por células do SI. Já a imunidade adaptativa humoral ocorre quando patógenos são eliminados por meio de moléculas secretadas pelo SI.

2.1.5 Imunidade adaptativa humoral

A imunidade adaptativa humoral atuam na resposta às infecções produzidas por patógenos extracelulares e na eliminação das toxinas produzidas por estes patógenos. Como citado anteriormente, o linfócito B é a principal célula desta imunidade e se caracteriza por ser o único capaz de produzir anticorpo [Abbas et al., 2012].

A imunidade adaptativa humoral tem como principal atividade a criação de anticorpos. Os anticorpos são imunoglobulinas produzidas por linfócitos B e liberadas na circulação e nos fluidos que recobrem as mucosas, neutralizando e eliminando

micro-organismos e produtos microbianos presentes no sangue e no lúmen de diferentes órgãos. A despeito de sua revelância em termos da resposta imune, neste trabalho não serão abordados a criação e o exercício do anticorpo.

2.1.6 Imunidade adaptativa Celular

A imunidade adaptativa celular atua na resposta os patógenos intracelulares. Estes podem se tornar intracelulares de duas formas: a primeira quando se deixa ser fagocitado por um macrófago, burla os mecanismos internos e escapa para o interior do citoplasma; a segunda forma é quando um patógeno penetra em uma célula e tem êxito infectando a mesma [Abbas et al., 2012].

Abbas et al. [2012] declara que a principal célula responsável pela imunidade adaptativa celular é o linfócito T. Este tem sua ativação dividida em fases distintas como: reconhecimento dos antígenos apresentados pelos APCs; ativação; expansão clonal e diferenciação. A Figura 2.6 apresenta este comportamento.

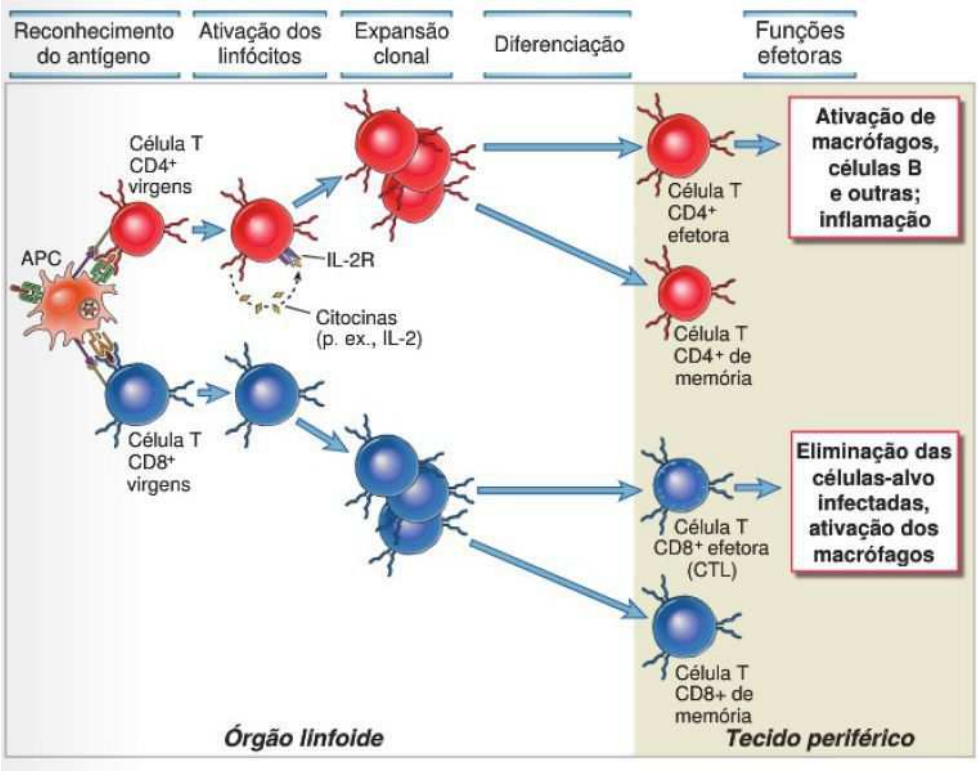

Figura 2.6. Fases das respostas da célula T.

Fonte: [Abbas et al., 2012]

A fase de reconhecimento consiste na apresentação de antígeno ao linfócito T no linfonodo ativando-o. Isto é feito por uma célula APC ativa. Segundo Abbas

et al. [2012], as células Th reconhecem antígenos apresentados pelo MHC-II; já as células CTL reconhecem antígenos apresentados pelo MHC-I. De acordo com este mesmo autor, as células APC apresentam antígenos no contexto MHC-I e MHC-II. Elas são dotadas da capacidade de fagocitar células infectadas, ao passo que extraem antígeno do patógeno invasor para ser apresentado.

A expansão clonal se inicia quando o linfócito é ativado. Consiste no aumento do número de células específicas para responder ao antígeno alvo. Neste momento, o linfócito passa de célula de reconhecimento de antígenos para célula efetora e ativa seus mecanismos de resposta. Com exceção dos linfócitos que ficam no linfonodo, os demais migram para o sítio da infecção [Abbas et al., 2012]. Em se tratando de mecanismo efetor, os linfócitos T e CTL se diferem.

O linfócito Th tem com principal ofício o provimento de estímulos para as demais células. Este estímulo se dá dentre outras formas como a liberação de citocinas. Tanto os macrófagos com as células APCs são estimuladas. Já o linfócito B é estimulado na criação de anticorpo. Como mencionado na seção 2.1.5, a produção de anticorpo não será estudada neste trabalho.

O linfócito T CD8+ ativado atua na produção de substâncias que são capazes de fagocitar outras células. Estes linfócitos se deslocam para regiões onde existem infecções, procuram células que tenham afinidade e que expressam antígenos por meio do MHC-I. Em caso de encontrar com células que atendam aos quesitos acima, o linfócito libera substâncias levando à célula infectada a fagocitose.

Como citado na seção 2.1.4, alguns linfócitos ativos que estavam participando do resposta, diferenciam-se e transformam-se em células T de memória. Estas sobrevivem no organismo por muito tempo. Este tipo de célula pode ser encontrado nos tecidos linfoides e nas mucosas. Todavia estas células não produzem mais citocinas nem fagocitam células infectadas. Portanto, células de memória se dividem em dois grupos sendo o primeiro nomeado de células de memória central, habitam nos órgãos linfoides e possuem capacidade de desencadear uma expansão clonal, caso encontre antígenos que a originaram. Já o outro grupo vive na mucosa e são chamadas de células de memória efetoras, estas possuem a capacidade de reagir a antígenos que reaparecerem nesta região [Abbas et al., 2012].

2.1.7 Autoimunidade

Como supracitado, o SI em funcionamento normal é capaz de reagir a uma quantidade imensurável de antígenos, tendo a habilidade de diferenciar o que é próprio do seu sistema ou autoantígenos, reagindo apenas ao não- próprio. Este processo

denomina-se tolerância imunológica que é composta pela tolerância central e periférica [Abbas et al., 2012].

No entanto, este sistema pode apresentar algumas falhas devido a fatores como distúrbios genéticos ou fatores ambientais no caso das infecções e ativar linfócitos auto-reativos. Os mesmos respondem as células do próprio corpo, e esta ação é chamada de autoimunidade. Assim, Abbas et al. [2012] declara: "autoimunidade resulta de uma falha dos mecanismos de autotolerância em células T ou B, o que pode levar a um desequilíbrio entre a ativação de linfócitos e os mecanismos de controle". Este mesmo autor afirma que nos EUA cerca de 2% a 5% da população sofrem com doenças causadas pela autoimunidade. Já Murphy [2010] firma que nos países ocidentais este percentual fica em torno de 5%. Na Figura 2.7, este autor apresenta algumas doenças que tem fenômenos de autoimunidade em sua patogênese. Na figura, constam o nome da doença, o mecanismo utilizado e a consequência.

2.2 Sistemas Multiagentes

Segundo Bordini et al. [2001], Sistema Multiagente (SMA) é classificado como uma área de pesquisa dentro da Inteligência Artificial Distribuída (IAD). A área de SMA tem como foco o estudo e a geração das soluções dos problemas relacionados à computação distribuída dentro da inteligência artificial. Segundo Weiss [1999], para solucionar os problemas distribuídos, o SMA tem em sua composição um grupo de elementos que são chamados de agentes.

Os agentes podem ter diferentes níveis de inteligência. O mais básico é quando o mesmo verifica o ambiente e executa a ação que lhe foi designada, em um nível mais sofisticado é exigido que o agente raciocine antes de executar a ação. Sobre as várias atividades desempenhadas pelos agentes, pode-se destacar a comunicação e o processo de mudança de estado do ambiente [Weiss, 1999].

Em um SMA, a cooperação é mútua e esta característica lhe atribui um poder computacional relativo à soma da capacidade de cada agente, ou seja, a união dos agentes resolve um problema que está além da capacidade individual de um agente. A área de SMA tem recebido grande atenção, uma vez que a Humanidade evolui rapidamente e juntamente com esta evolução os problemas se tornam maiores e mais complexos, dinâmicos e distribuídos. Devido a esta mudança de cenário, algumas soluções não atendem mais e devem ser remodeladas com o intuito de se enquadrarem nos padrões impostos pelo crescimento. Padrões estes que para serem

Doença	Mecanismo da doença	Consequência
Doença de Graves	Autoanticorpos contra o receptor do hormônio estimulante da tireoide	Hipertireoidismo: superprodução de hormônios da tireoide
Artrite reumatoide	Células T atacam antígenos da sinóvia da articulação	Inflamação e destruição das articulações, causando artrite
Tireoidite de Hashimoto	Autoanticorpos e células T autorreativas contra antígenos da tireoide	Destruição de tecidos da tireoide causando hipotireoidismo: subprodução de hormônios da tireoide
Diabetes tipo I (Diabetes melito insullino--dependente, IDDM)	Células T autorreativas contra antígenos das células pancreáticas presentes nas ilhotas	Destruição das células β pancreáticas levando à não-produção de insulina
Esclerose múltipla	Células T autorreativas contra antígenos do cérebro	Formação de placas escleróticas no cérebro com destruição das bainhas de mielina que envolvem os axônios na célula nervosa, levando à fraqueza muscular, à ataxia e a outros sintomas
Lúpus eritematoso sistêmico	Autoanticorpos e células T autorreativas contra DNA, cromatina, proteínas e antígenos de ribonucleoproteínas ubíquos	Glomerulonefrite, vasculite, eritema
Síndrome de Sjögren	Autoanticorpos e células T autorreativas contra antígenos de ribonucleoproteínas	Infiltração de linfócitos em glândulas exócrinas, levando a ressecamento dos olhos e/ou boca; outros órgãos podem estar envolvidos, levando à doença sistêmica

Figura 2.7. Algumas doenças autoimunes.

Fonte: [Murphy, 2010]

atendidos é necessário utilizar técnica que seja capaz de agir de forma distribuída e autônoma, diante de problemas dinâmicos e complexos. O SMA tem eficiência no desenvolvimento de modelos e teorias. Devido a este fato, vêm sendo utilizado cada vez mais no mundo das pesquisas científicas. Ele está relacionado com o entendimento dos agentes juntamente com os seus tipos e do ambiente que os agentes atuam [Weiss, 1999].

2.2.1 Ambiente

O ambiente pode ser entendido como sendo o mundo em que o agente se encontra, neste mundo ele pode interagir com outros agentes e com o próprio ambiente, possibilitando-o através de ação mudar o ambiente. Wooldridge [2001] declara que a natureza e a característica do ambiente podem variar e classifica os ambientes como:

- **Acessível × Inacessível:** neste ambiente é possível ter acesso às informações completas, precisas e atuais sobre o seu estado, esta característica não é comum à maioria dos ambientes. Um exemplo é o espaço quando se deseja obter a posição de uma estrela, esta informação tem um atraso na escala de anos-luz;

- **Determinístico × Não-determinístico:** a característica do ambiente determinístico é que uma ação é absoluta, ou seja, para cada ação existe um único resultado, pode se dizer que o próximo estado do ambiente é determinado pelo estado atual e pela ação do agente. Já o não determinístico é quando o agente não consegue prever o próximo estado do ambiente, bem como o comportamento dos outros agentes.

- **Estático × Dinâmico:** O ambiente estático tem a característica de não mudar o seu estado a não ser que um agente o mude. Já o ambiente dinâmico o estado pode ser alterado independente da ação do agente, nesta abordagem o ambiente tem outros procedimentos que alteram seu estado.

- **Discreto × Contínuo:** Pode-se dizer que um ambiente é discreto se existe um número finito de estados que ele possa assumir, mesmo que este número seja muito grande. Já um ambiente contínuo pode assumir um número infinito de estados.

2.2.2 Agentes

De acordo com Bordini et al. [2001] não existe um consenso referente à definição de agente entre os pesquisadores da área de IA. Na ausência desta definição e como existem muitas sugestões, seguem algumas: "um agente é um sistema computacional, situado em um ambiente, que é a capaz de tomar ações autônomas nesse ambiente com a finalidade de atingir os objetivos para os quais fora construído"[Wooldridge, 2001], um outro conceito relevante "significa o fato de um agente ter sua existência independente dos demais, e até mesmo do problema a ser solucionado"Hübner et al., 2004 e "para funcionar, um agente não precisa de outros agentes, mesmo que para alcançar seus objetivos ele eventualmente vá precisar da ajuda de outros."Hübner et al., 2004.

No entanto, Bordini et al. [2001] apresenta alguns aspectos importantes na tentativa de facilitar a compreensão do que é um agente em um contexto de SMA. Estes aspectos não têm a intenção de gerar uma definição universal do termo agente, mas sim apresentar uma concepção específica de agente com base na experiência dos

autores, sendo assim um agente neste contexto é um sistema computacional capaz de:

- **percepcão:** através dos sensores, o agente é capaz de perceber alterações no ambiente;
- **ação:** a execução de ações no ambiente faz com que o mesmo seja alterado, o objetivo do agente (motivação) é alcançado através de uma ou mais ações no ambiente alterando-o para o estado em que o agente deseja. Estas ações são realizadas através dos efetuadores do agente;
- **comunicação:** a comunicação com outro agente da sociedade também é considerada uma ação que o agente realiza, devido ao fato de que em algumas situações é necessário coordenar atividades e é neste momento que é realizada a ação de comunicação entre os agentes;
- **representação:** o agente tem consigo uma representação do que ele acredita ser verdade sobre o ambiente e os outros agentes. Esta representação é simbólica e explicitada;
- **motivação:** em SMA os agentes podem ser modelados como autônomos e têm acesso à representação do conhecimento, desejos, objetivos. Diante deste fato, o agente conhece o estado do ambiente que ele deseja alcançar, então ele pode, por si próprio, agir sobre o ambiente para atingir a meta ou satisfazer os objetivos;
- **deliberação:** de posse das informações necessárias(motivação, representação do estado, do ambiente) o agente tem a capacidade de predizer possíveis estados para o ambiente e executar ações que satisfazem os seus objetivos. A satisfação de um objetivo do agente significa que o ambiente está como desejado;
- **raciocínio e aprendizagem:** a utilização de técnicas de inteligência artificial clássica em agentes, por exemplo, no raciocínio e na aprendizagem é uma forma que pode melhorar o desempenho de SMA quando estendida para múltiplos agentes. Com relação à deliberação, nem sempre se espera que todos os agentes tenham esta função, uma vez que não é uma tarefa simples. A criação de um mecanismo de aprendizagem para ambientes multiagentes é uma área de pesquisa que requer muita investigação.

Na Figura 2.8 é apresentado um modelo de um agente, e mostra o ciclo das atividades que este realiza para que uma ação seja refletida no ambiente. O agente percebe o ambiente através do sensor, processa seu conjunto de regras e verifica a validade dos desejos, executando assim por meio dos efetuadores uma ação no ambiente. Para verificar se o desejo foi cumprido, o sensor do ambiente é acionado novamente recebendo assim um *feedback*.

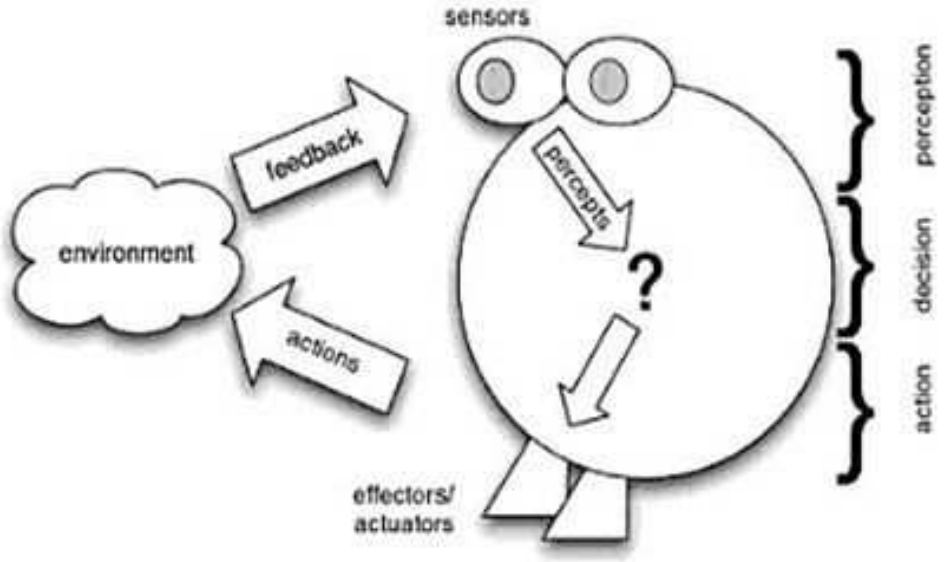

Figura 2.8. Um agente no seu ambiente.

Fonte: [Wooldridge, 2001]

2.2.2.1 Agentes reativos

Nesta arquitetura a ideia é que a inteligência não está no agente e sim nas várias interações que são realizadas. Tais interações dão origem a um comportamento ordenado inteligente e sincronizado. Esta abordagem tem grande influência da entologia (estudos dos insetos) [Hübner et al., 2004]. Como exemplo de um SMA que a inteligência emerge do coletivo, pode-se citar a colônia de formiga, cada uma tem uma função limitada que, quando observado o coletivo, percebe-se o comportamento inteligente.

Os agentes reativos não conhecem explicitamente o ambiente com o seu estado e os outros agentes, nem tampouco guardam um histórico das ações anteriores. Estas características possibilitam que agentes deste tipo, possam ser representados por um autômato finito simples, uma vez que o seu conjunto de regras mapeiam as percepções do ambiente direto em ações (estímulo/resposta) [Hübner et al., 2004].

2.2.2.2 Agentes cognitivos/deliberativos

Este tipo de agente desempenha tarefa mais complexas por ter conhecimento do ambiente e dos outros agentes e é capaz de raciocinar antes de agir. Então, para que uma ação seja executada, o agente realiza a leitura do ambiente e a verificação do plano de ação. De posse destas informações, ele verifica qual a melhor ação a ser executada para atingir a meta.

Os SMAs com agentes cognitivos/deliberativos se diferem dos reativos, pois possuem poucos agentes, devido ao fato de que cada agente é um sistema complexo [Bordini et al., 2001]. Como estes agentes têm uma complexidade maior, para que tudo funcione é necessário ter uma arquitetura que suporte estes recursos. Segundo os autores Wooldridge [2001]; Hübner et al. [2004], a arquitetura mais difundida para desenvolver SMA com agentes cognitivos/deliberativos é a arquitetura BDI, pois nesta é possível implementar as crenças, desejos e intenções do Inglês *belief, desire e intention*.

2.2.3 Multiagente paralelo

O desafio que a computação enfrenta é no sentido de avaliar, modificar, compor, gerenciar e explorar modelos computacionais para vários domínios de aplicação. Isto leva em consideração todo o volume de informações geradas e até mesmo o tempo de resposta dos experimentos.

No entanto, a quantidade de dados processados durante a simulação dos experimentos *in-silico* está ficando cada vez maior e as operações executadas sobre estes dados mais complexas. Devido a este fato, os sistemas de simulação deverão ser reformulados para atender a esta nova demanda. Diante deste cenário, é factível a utilização de *frameworks* HPC [Collier & North, 2012].

Zia et al. [2012] declara que existem diversos *frameworks open source* multiagente para simulação, mas a maioria não dá suporte ao desenvolvimento paralelo HPC. Este fato inviabiliza a utilização dos mesmos, uma vez que o tempo de resposta da simulação e quantidade de dados analisados são grandes.

Como citado anteriormente, as simulações estão exigindo cada vez mais desempenho, e na simulação do sistema celular (bioinformática) não é diferente, pois demanda um alto poder computacional. Então é adequado que se utilize um modelo baseado em agentes e que funcione em uma arquitetura paralela. Esta arquitetura pode rodar tanto em um *cluster* de alto desempenho ou em GPU [Richmond et al., 2010]. Li et al. [2009] também indica o uso de *cluster* para simulações que utilizam agentes reativos.

De acordo com Sousa et al. [2014], novas funcionalidades estão sendo adicionadas ao *AutoSimmune*. Este fato torna o sistema mais complexo e abre novas possibilidade para testar conjuntos maiores de dados. Portanto, para que os experimentos satisfaçam os requisitos de tempo e quantidade de informações processadas, é aconselhada a programação paralela. De acordo com Possi [2012], o *AutoSimmune* deve evoluir para uma arquitetura de processamento paralelo, ou seja, que se utilize um *framework* HPC.

2.3 GPGPU

Unidade de processamento gráfico (GPU) do inglês (*Graphics Processing Units*), foi amplamente difundida por ser utilizada em jogos na renderização e exibição de imagens. Pois, estes dispositivos possuem uma grande capacidade de cálculo de ponto flutuante em paralelo. Tais dispositivos eram apenas aceleradores gráficos e suportavam somente pipeline de funções fixas específicas [NVIDIA, 2015].

No entanto, alguns pesquisadores viram a possibilidade de tirar proveito e começaram a modelar seus problemas para serem executados em placas gráficas e aproveitar a performance de cálculo de ponto flutuante. Então a GPU tornou-se cada vez mais programável. Em 1999, a NVIDIA iniciou a produção deste hardware, e algum tempo depois com a inclusão de mais recursos para programação de propósito geral nasceu a (GPGPU), que seria a GPU de propósito geral [NVIDIA, 2015].

A GPU é extremamente mais rápida no processamento de cálculo de ponto flutuante, visto que foi projetada para atender o uso intensivo de computação paralela. Pois possui mais transistores que são dedicados ao processamento, em vez de controle de fluxo. A Figura 2.9 mostra a diferença entre a CPU e GPU.

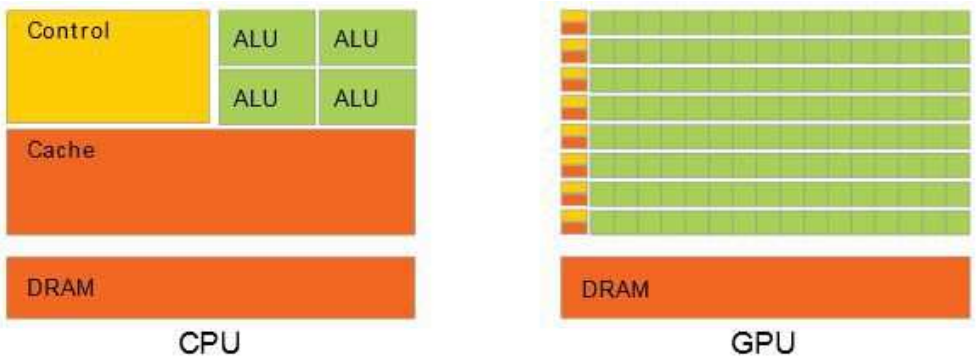

Figura 2.9. The GPU Devotes More Transistors to Data Processing.

Fonte: [NVIDIA, 2015]

A GPU implementa um modelo computacional que possui vários processadores, porém com apenas uma unidade de processamento. O modelo computacional é chamado instrução única, dados múltiplos do inglês *Single Instruction Multiple Data* (SIMD). Portanto, esta abordagem realiza muitas operações em paralelo, mas sempre a mesma operação [Pereira, 2011].

2.3.1 Modelo de programação

A programação para placa gráfica é composta de uma parte do código executado na CPU e outra na GPU. O código CUDA a ser envido para GPU, pode ser escrito de diversas formas, uma vez que a plataforma foi projetada para suportar a linguagem de programação C como sendo a principal de alto nível. Outras linguagens como C++, *Fortran*, Java, *Python*, podem ser utilizadas através de interfaces. Também existe possibilidade de desenvolver com *directives*, é o caso do *OpenACC*; além de outras opções como *Thrust* e etc.

A Figura 2.10 apresenta um exemplo de soma de matrizes, onde o resultado de A[i] mais B[i], estão sendo armazenados em C[i]. No exemplo, é apresentado o método **VecAdd** que será executado na placa gráfica. Este método é chamado de *kernel*. Ele é executado inúmeras vezes pelas *threads* e possui em sua assinatura a palavra "***global***".

```
// Kernel definition
__global__ void VecAdd(float* A, float* B, float* C)
{
    int i = threadIdx.x;
    C[i] = A[i] + B[i];
}

int main()
{
    ...
    // Kernel invocation with N threads
    VecAdd<<<1, N>>>(A, B, C);
    ...
}
```

Figura 2.10. Código de um *Kernel* que soma duas matrizes.

Fonte: [NVIDIA, 2015]

Métodos que são executados na placa gráfica não retornam valores. A troca de informação entre CPU e GPU é realizada pelo comando **cudaMemcpy**. Este comando possibilita o envio das informações para GPU e retorno do resultado para CPU.

As *threads* na arquitetura CUDA são organizadas em uma hierarquia de blocos e grades. A Figura 2.11 apresenta um modelo desta hierarquia. Um **SM** executa as *threads* de um bloco em grupos de 32 *warp*, ou seja, o número de *threads* por bloco tem que ser múltiplo do tamanho do *warp*. De acordo com NVIDIA [2015], cada bloco possui no máximo 1024 *threads*.

A Figura 2.11 aponta um **GRID** com dimensão 3 x 2 contendo 6 blocos. Cada bloco 2D com dimensões 4 x 3 com 12 *threads* cada.

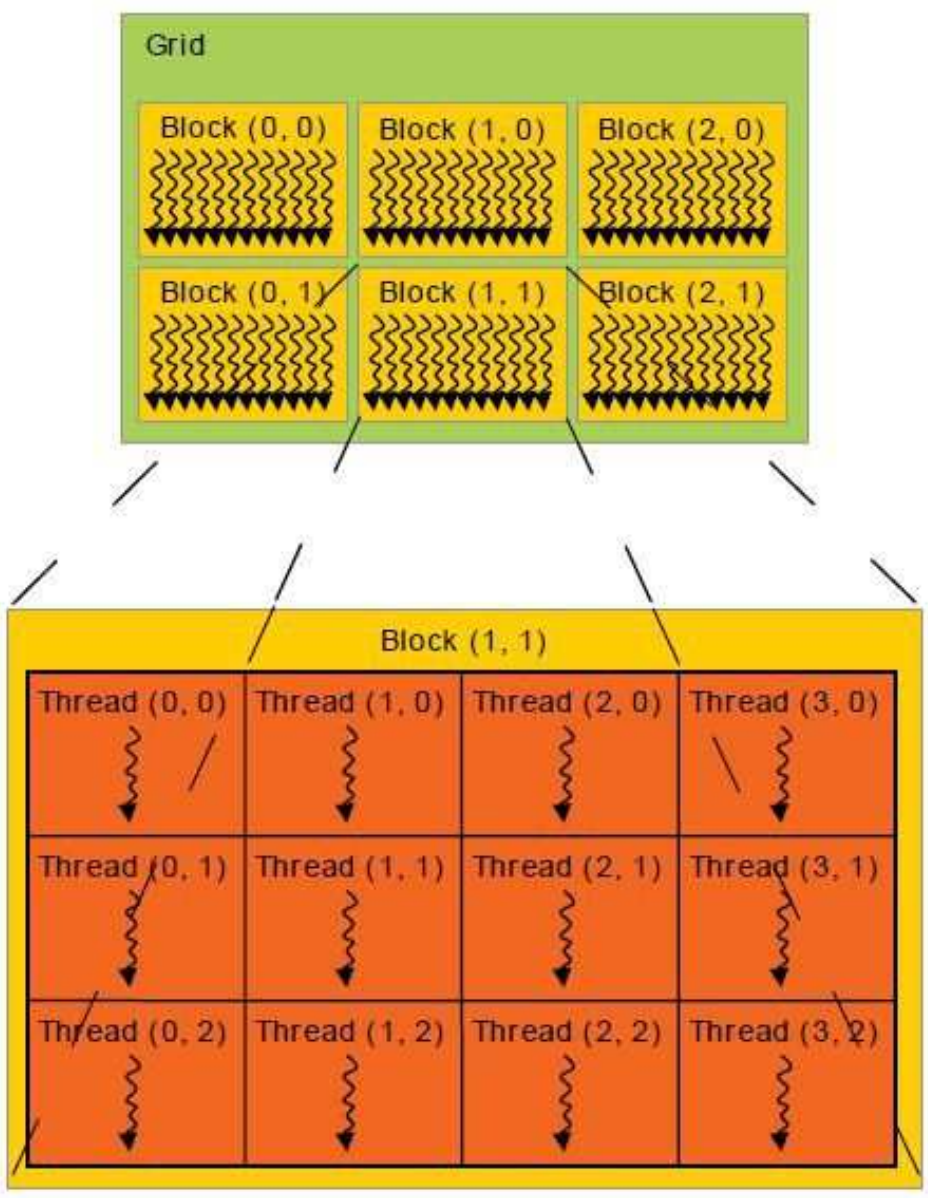

Figura 2.11. Grid of Thread Block.

Fonte: [NVIDIA, 2015]

2.4 Frameworks

No desenvolvimento de *Agent-based modeling and simulation* (ABMS) é indispensável a utilização de *framework*, pois este disponibiliza um conjunto de funcionalidades, isto é um *tookit*, específico para cada domínio de aplicação, que tem a finalidade de auxiliar a implementação das funcionalidades [Macal & North, 2009]. Possi [2012] apresenta em seu trabalho um estudo e descreve alguns *frameworks*. Mas define o *Repast Simphony* para ser utilizado na implementação da primeira versão do *AutoSimmune*.

Collier & North [2012], diz que ABMS que não seja HPC deve ser remodelado e atualizado. E apresenta o *Repast* HPC, que é o *framework* a ser utilizado, para atender as novas demandas computacionais, com possibilidade de ser executado em *cluster*. Ainda segundo este autor os ABMS tradicionais deverão ser utilizados apenas em problemas simples.

Portanto, para definição do *framework* a ser utilizado neste trabalho, foi consultado o trabalho de [Possi, 2012]. Além disso, foi investigado o *Repast* HPC que tem como característica a movimentação do agente por zonas. Assim sendo, em uma movimentação onde o agente passasse para outra zona o mesmo, é removido e é criado um novo na outra zona. Como a simulação do SI demanda muita movimentação dos agentes, tornou-se inviável utilizar o *Repast* HPC.

Já o Flame GPU tem suas características de modelagem idênticas ao Flame que será citado abaixo. Porém toda simulação é realizada na GPU e como foi citado anteriormente a GPU é ideal para resolver cálculos, ou seja, trabalhar com dados lineares. Segundo Richmond et al. [2010], simulações com o Flame GPU devem observar algumas restrições, tais como possuir um grande número de agentes relativamente simples, uso de poucas variáveis por agente, e pouca troca de mensagens. Estas restrições limitam as simulações relacionadas ao sistema imune humano. **Portanto, neste trabalho será utilizado o *framework* Flame**, cuja exposição e razões para escolha serão apresentadas a seguir.

2.4.1 Flame

Flame é um *framework* flexível que foi idealizado para funcionar em arquitetura paralela (*cluster*). Composto de uma linguagem para especificação de agentes e uma ferramenta de conversão do modelo para o código, que baseado em templates gera um código otimizado, eficiente e pode ser executado tanto em arquitetura serial como paralela [Worth et al., 2012]. Portanto, os agentes são gerados automaticamente e

compilados na linguagem de programação C. Logo, este *framework* se enquadra na geração HPC.

O processo de desenvolvimento neste *framework* é divido em fases como: definição do modelo em um XML, codificação das funções em código C, associação das duas etapas anteriores com os templetes da ferramenta através do componente *xparcer*.

O XML descrever o modelo e a sequência de execução das operações. Já o arquivo C contém as funções que o agente irá realizar. O processo de *parcer* (*xparcer*) tem como entrada estas informações e gera a aplicação. A Figura 2.12 mostra os passos para gerar uma aplicação no Flame.

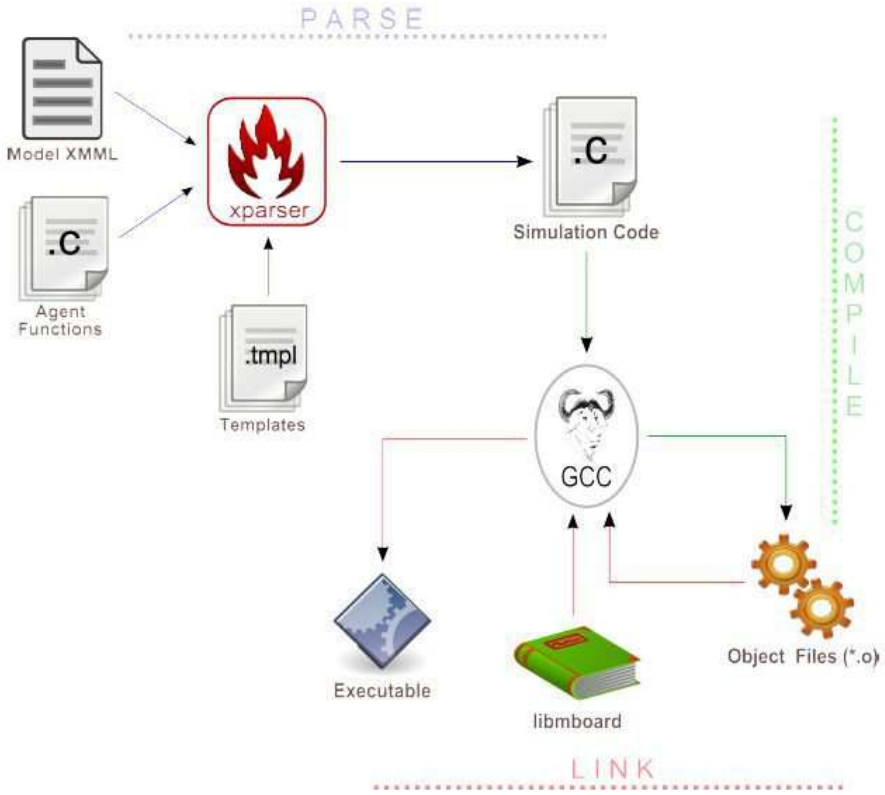

Figura 2.12. The structure of the FLAME Environment.

Fonte: [Worth et al., 2012]

Em um ABMS, existem funções básicas como mudança de estado dos agentes e a comunicação entre os mesmos. No Flame, estas atividades funcionam tendo como suporte caixa de mensagem (*message boards*). Estas caixas possuem um tipo único de mensagem que pode ser composta por mais de um campo como: identificador e conteúdo. Contudo, em uma caixa não pode ser adicionado outro tipo [Worth et al., 2012].

Os fatos citados acima tornam a leitura e a escrita das mensagens simples e possibilitam que os agentes sejam divididos em várias áreas da memória. Portanto, estas características permitem que programas desenvolvidos com o Flame funcionem de forma serial ou paralela [Worth et al., 2012].

Todavia, quando um agente necessita comunicar com outro, o mesmo coloca uma mensagem na caixa para que o outro leia. O Flame gerencia todas as caixas de mensagens utilizando a API *Message Board API* de forma simples, porque que as caixas possuem um tipo único e as operações (leitura/escrita) sobre elas são simples. A Figura 2.13 ilustra o comportamento descrito acima [Worth et al., 2012].

No entanto Chin et al. [2012] declara que o Flame utiliza estratégias diferentes de acesso às caixas de mensagens. Sendo que da forma serial todos os agentes acessam a mesma caixa, com o controle de concorrência centralizado. Já na forma paralela, cada agente possui sua caixa de mensagem e o Flame promove a sincronização das mesmas. Ainda seguindo o autor, todas as operações sobre as caixas são realizadas utilizando uma biblioteca chamada *libmboard*. A Figura 2.13 ilustra o comportamento descrito acima

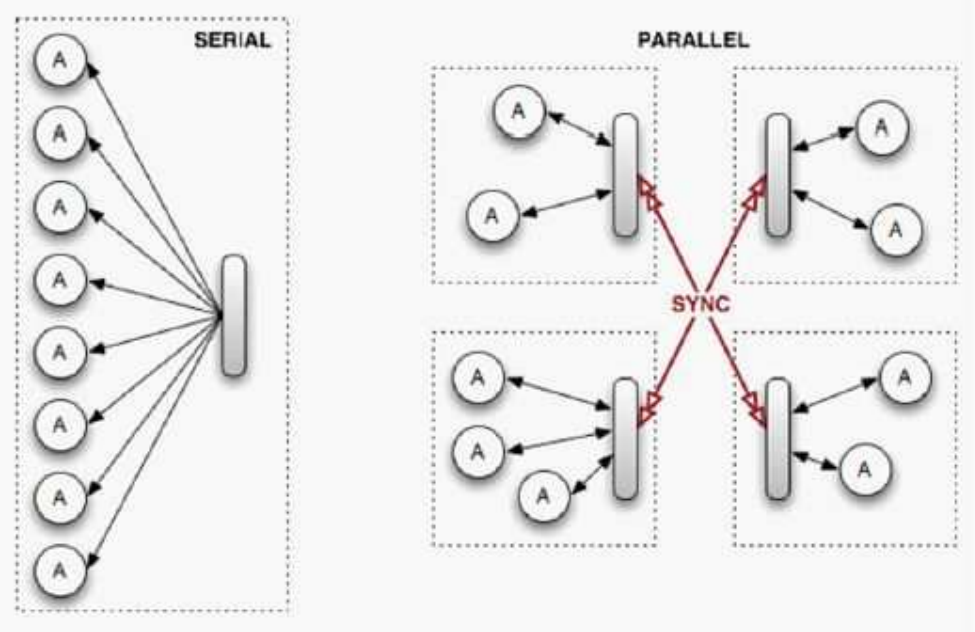

Figura 2.13. Serial and Parallel Message Boards.

Fonte: [Chin et al., 2012]

Como já apontado, não é aconselhável utilizar o Flame GPU para simulação do SI. No entanto, foi identificada uma funcionalidade ideal para ser processada na GPU. Então, se fez necessário integrar o *framework* Flame a GPU. Desta forma,

o agente e sua movimentação ficaram na incumbência da CPU. Já o processo de difusão de substância foi direcionado para GPU fazer o processamento.

2.5 Trabalhos Correlatos

Possi [2012] apresentou alguns trabalhos que guiaram sua pesquisa. Porém, aqui será citado apenas uma descrição resumida dos mais relevantes para este estudo, visto que a referência principal deste é o *AutoSimmune*.

Simmune, de acordo com Rapin et al. [2010], é considerada uma técnica de desenvolvimento juntamente com uma linguagem, ou seja, é uma plataforma para simulação do SI. Este ambiente funciona como uma rede tridimensional onde todos os elementos do SI são partículas (exemplo: as citocinas). No entanto, as células são modeladas como agentes reativos puros e têm como função apenas reagirem em respostas a estímulos.

***The Basic Immune Simulator* (BIS)**, apresentado por Folcik et al. [2007], foi criado inicialmente para modelar as células da imunidade inata e interação com as células da imunidade adaptativa. Todos os comportamentos acontecem nos três espaços que o modelo disponibiliza. Conquanto, Rapin et al. [2010] declara que o BIS está sendo usado em ambientes educacionais e em pesquisas que estudam temas como hipersensibilidade e câncer.

AutoSimmune simulador do SI desenvolvido pelo grupo de pesquisa da Universidade Federal de Viçosa envolvendo o Departamento de Informática e o Departamento de Medicina e Enfermagem, busca a simulação do sistema imune por meio da programação orientada a agentes. Este simulador foi aplicado em diversos fenômenos biológicos relacionados com o sistema SI, tais como o desenvolvimento da autoimunidade [Possi, 2012]; verificação do papel do mastócito em infecções [Silva et al., 2012]; estudo da resposta imune na glomerulonefrite pós-infecciosa (GnPE) por *Streptococcus pyogenes* [Bastos et al., 2013]; verificação do papel dos Linfócitos T CD4+CD25+ na regulação do sistema imunológico sob a perspectiva ocorrência da sepse [Siqueira-Batista et al., 2010] e a simulação da imunologia da sepse [Sousa et al., 2014]. O *AutoSimmune* foi desenvolvido sobre o framework **Repast Symphony** que utiliza a linguagem de programação Java. Segundo Possi [2012] o *AutoSimmune* foi desenvolvido baseado no BIS.

Capítulo 3

Proposta de modelo do SI humano para simulação da autoimunidade

Neste capítulo será descrita a proposta do modelo para simulação do SI. Este guiará a implementação da ferramenta que realizará as simulações *in-silico*. Assim como Possi [2012] apresentou os itens do modelo, este trabalho seguirá praticamente a mesma estrutura apresentada pelo autor supracitado. Isto para facilitar a comparação e a percepção das diferenças entre os modelos.

3.1 Requisitos do modelo

Este tem como propósito o estudo da autoimunidade. Portanto, os aspectos a serem observados são exatamente os que geraram doenças autoimunes. No entanto, os mecanismos que podem falhar e gerar estas doenças são nomeados de tolerância imunológica, que se divide em tolerância central e tolerância periférica.

A tolerância central é caracterizada por possuir duas atividades distintas: a primeira chamada de seleção positiva é a atividade que elimina os linfócitos não funcionais; a segunda atua na eliminação dos linfócitos auto reativados e é chamada de seleção negativa. Do mesmo modo que Possi [2012], neste estudo será considerada apenas a seleção positiva com relação à tolerância central.

Na literatura pesquisada a tolerância periférica é caracterizada por desempenhar diversas atividades. No entanto, o que mais se destaca é a necessidade da existência de um sinal para indicar ao linfócito se o antígeno é perigoso. Como citado na seção 2.1.4, este sinal é chamado de *danger signal*.

Além dos mecanismos citados acima, é necessário uma funcionalidade para geração de diversidade. Esta tem como finalidade a geração de diversas especificidades

utilizadas pelo modelo. Para isto, é necessário um banco de dados de antígenos próprios e autoantígenos a serem utilizados durante a tolerância central.

Até o momento só foram citados mecanismos para reconhecimento de antígenos através de seus receptores. Porém, faz-se necessário modelar o procedimento de afinidade, quer dizer, a força com que o antígeno é reconhecido. A vista disso é importante ressaltar que também faz parte do modelo o mecanismo para reconhecimento de antígenos e o cálculo da afinidade.

3.2 Framework alvo

Este modelo será implementado utilizando o *framework* Flame em conjunto com as bibliotecas *xparcer* versão 0.17.0 e a biblioteca *libmboard* versão 0.3.1. O trabalho foi desenvolvido na plataforma Linux (Ubuntu 14.04). No entanto, como utiliza a linguagem de programação C é possível portar esta aplicação para outras plataformas como Windows.

Este *framework* disponibiliza um recurso que permite a definição da quantidade de *ticks* a serem executados. Com relação a representações de células, tecidos, substâncias entre outros, ele não disponibiliza recursos para facilitar a implementação. Ficando a cargo do desenvolvedor definir as estruturas a serem utilizadas.

O modelo descrito neste trabalho não depende de *framework* ou plataforma, podendo ser adaptado para outras situações, uma vez que os conceitos de imunologia não estão ligados às características do Flame.

3.3 Ambiente

As características do ambiente estão ligadas diretamente com a forma que o SI foi modelado, portanto as perspectivas dos agentes podem ser classificadas em: inacessível, não-determinístico, dinâmico e discreto.

- **Inacessível:** os agentes participantes das simulações não têm informações do ambiente como um todo. Logo eles trabalham com informações locais, e quando necessário, é solicitado ao ambiente a informação de algum vizinho. A comunicação entre os agentes no Flame é através de caixas de mensagens. Como apresentado na seção 2.4.1.

- **Não-determinístico:** não existe a possibilidade de um agente predizer o próximo estado do ambiente, bem como o comportamento de outros agentes

e o comportamento da simulação como difusão de substância.

- **Dinâmico:** a mudança de estado do ambiente não depende da tomada de decisão de algum agente, porque independente da ação do agente o ambiente pode ser modificado.

- **Discreto:** nas simulações existem muitos agentes, mas é possível saber as possíveis posições que um agente pode ocupar, já que o mesmo não se movimenta em um espaço infinito. Com relação às substâncias, existe um espaço definido para difusão das mesmas. Portanto, estas características classificam este modelo como um ambiente discreto.

Neste trabalho, foi necessário definir um agente para conter algumas estruturas de dados e métodos comuns a todos os outros. As estruturas são as zonas físicas que julgamos não triviais para serem colocadas em caixa de mensagem, pois são usadas para difusão de substâncias. Então foi criado um agente que representa o ambiente. Este possui diversas funções como:

- Disponibilizar todos os métodos comuns aos agentes, como:

  int isEspecificNeighborsRadius(int analyzeX, int analyzeY, int x, int y, int radius), verifica se as posições *analizeX* e *analyzeY* são vizinhas das posições x e y, obedecendo o parâmetro informado como raio (*radius*).

  int match(char* caracter1, char* caracter2), usado para calcular a afinidade, onde são informadas duas cadeias de *bits* e a função retorna 1 sucesso e 0 caso contrário. Internamente utiliza o parâmetro AFFINITYTHRESHOLD para decidir o retorno.

- agrupar a estrutura de dados contendo as camadas ou zonas e realizar a difusão de substâncias.

- inserir os vírus na simulação.

Neste modelo a camada de dados ficará associada ao agente que representa o ambiente. Outros agentes podem realizar dois tipos de tarefa sobre a camada de dados. Uma é a liberação de substância que é realizada através do método **void releaseCitokineLymphnode(int x, int y, int zoneCitokineLymphnode)**. Este método tem como parâmetros: as coordenadas x, y, onde será liberada a substância; e a camada da substância *zoneCitokineLymphnode*. Em cada camada é liberado um

tipo específico de sustância. O valor da concentração a ser liberada é obtido pelo parâmetro referente à substância.

Com relação à estrutura de dados (zonas e camadas de substâncias), apenas o agente representando o ambiente acessa diretamente. Os demais agentes realizam este acesso utilizando o recurso da linguagem de programação C. Isto foi possível devido ao fato de que o Flame foi desenvolvido nesta linguagem.

Para que um arquivo .c acesse métodos de outro arquivo .c, simulando uma herança em orientação a objetos, é necessário incluir no cabeçalho do arquivo a assinatura do método que se deseja acessar e adicionar na frente da assinatura a directiva ***extern***.

Como exemplo no agente ***functionsmacrophage.c*** foi adicionado ***extern int isEspecificNeighbors(int analyzeX, int analyzeY, int x, int y)***. Isto indica que este método foi implementado em outro arquivo, ou seja no ambiente. Portanto todas as funções comuns foram implementadas somente no ambiente, e onde for necessário utilizar foi adotada a solução citada acima.

Assim como no trabalho de Possi [2012], este foi projetado contendo diversas zonas, mas somente um ambiente, podendo ser dividido em três agrupamentos que são:

- **Ambiente:** É responsável por gerenciar as difusões de substâncias e possui todas as camadas de dados do modelo e métodos comuns.
- **Zonas Reais:** São as zonas que ocupam um espaço na memória. Apesar de não serem visualizadas elas existem. É onde ocorrem as difusões de substâncias.
- **Zonas Virtuais:** São zonas que não possuem uma representação na memória. Exemplo as zonas que são usadas para delimitar a movimentação de um agente. Não é necessário existir fisicamente, pois o espaço pode ser controlado por parâmetros delimitadores. Todas as zonas serão discutidas posteriormente.

3.4 Tempo

Neste trabalho será utilizada a unidade de tempo discreta que Possi [2012] chamou de *tick*. De acordo com o mesmo autor, o *tick* é um intervalo de tempo gasto para que sejam executadas todas as atividades agendadas. Desta forma, o próximo *tick* só poderá ser executado quando todas as atividades do anterior forem finalizadas.

No Flame, este procedimento é executado automaticamente sem a necessidade de alguma informação extra, como anotações descritas por Possi [2012]. Foi mostrado na seção 2.4.1 que o Flame possui um arquivo XML com a descrição e ordem de execução das funções que se encontram em um arquivo **.c**. Portanto, o *framework* realiza a execução de todos os métodos do agente na ordem descrita no arquivo XML. Em casos onde existem vários agentes o *framework* executa um método de cada agente na ordem, ou seja, é executado o primeiro método do agente 1 e em seguida o primeiro método do agente 2, e assim por diante.

Portanto, um *tick* neste trabalho compreende a execução de todos os métodos de todos os agentes, seguindo a ordem citada acima. A execução de um *tick* compreende em alterar a posição dos agentes, liberar substâncias, analisar a vizinhança através de leitura da caixa de mensagem e realizar a difusão de substâncias.

O *tick* pode ser trabalhado de duas maneiras. A primeira diz respeito ao desempenho, ou seja, o tempo que a máquina levou para processar um *tick* ou todos os *ticks*. Esta visão está relacionada à otimização e é exatamente o que este trabalho comparará em relação a Possi et al. [2011]. No entanto, em uma simulação, é necessário que se faça uma abstração do *tick* com o tempo real. Isto possibilitará dizer quantos serão necessários para equivaler ao tempo real (uma hora, um dia, um mês, um ano e etc.).

Para rodar um experimento no Flame, a linha a ser executada na plataforma Linux é : **./main 3000 0.xml -p -f 50** Onde:

- **./** - comando do Linux para rodar uma programa.
- **main** - este é o binário gerado após a compilação pelo comando *make*
- **3000** - quantidade de vezes que o programa vai rodar (*tick*).
- **0.xml** - modelo representando os agentes.
- **-p** - indica execução em paralelo.
- **-f 50** - em cada execução, o Flame gera um arquivo XML com o resultado do *tick*. Este parâmetro indica que vai gerar um arquivo a cada 50 *ticks*.

Após analisar algumas execuções, observou-se que a escrita dos arquivos XML no disco representava um gargalo. Para resolver este problema, foi utilizado um comando do Linux que permite mapear parte da memória RAM para uma pasta. Ao escrever nesta pasta, o programa está escrevendo na memória principal. Então

foi criada uma pasta chamada *driver-virtual* e, ao executar o comando abaixo, esta é associada a aproximadamente 7GB da memória principal. Segue o comando:

sudo mount -t tmpfs none driver-virtual/ -o rw,size=7000m

Portanto todos arquivos XML da execução, foram direcionados para esta pasta, para direcionar basta copiar o 0.xml para dentro desta pasta e alterar o caminho no momento da execução (./main 3000 driver-virtual/0.xml ...). O comando utilizado para voltar a memória e a pasta ao normal é: **sudo umount driver-virtual/**

3.5 Espaço

O Flame gerencia o espaço ocupado pelo agente, ou seja, o *framework* aloca o agente em uma posição de memória que ele acha mais viável. O próprio *framework* possui mecanismos que não possibilita que um agente acesse os dados do outro diretamente, garantindo assim o isolamento entre os agentes. No *Repast Symphony* o agente é criado e adicionado em uma estrutura de dados chamada *DefaultContext*, no entanto em ambas as implementações de espaço o agente movimenta como se estivesse em um espaço toroidal.

3.5.1 Projeções

No Flame não existe uma estrutura pré-definida, mas a forma em que o agente comporta define a projeção, uma vez que ele está imerso em um estado discreto. Pois em sua criação e movimentação os valores das coordenadas cartesianas (x,y,z) 3D devem ficar entre os parâmetros de largura e altura do espaço definido para o parâmetro Z.

O parâmetro Z, consiste na ZONA em que o agente se encontra e cada zona tem um tamanho diferente. Então os valores para os atributos x e y do agente deverá ficar entre o intervalo definido pelos parâmetros *WIDTH* e *HEIGHT* de cada zona. O fato da informação da posição do agente (x, y, z) estar contida no próprio, descarta a existência de um *grid* 3D na memória. A Figura 3.1 ilustra o posicionamento de um agente.

Já as projeções contendo as substâncias possuem uma representação na memória, ou seja, um *grid* 2D. Esta representação consiste em armazenar o valor da concentração da substância em um determinado ponto (x,y). Então quando se deseja saber o valor de uma substância em uma determinada ZONA, que não seja virtual. Basta utilizar a função (***float getCitokineValue*(int x, int y, int**

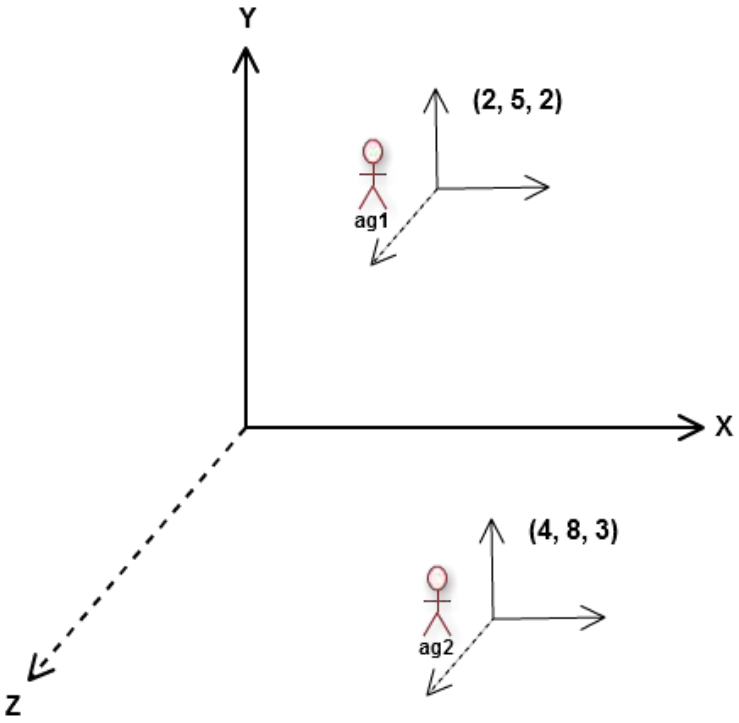

Figura 3.1. Posição de um agente no espaço 3D.

zoneCitokine)). Esta função retorna o valor de uma substância presente nas coordenadas x,y da zona Tissue, porém a projeção da substância é indicada pelo valor do parâmetro *zoneCitokine*.

A Figura 3.2 apresenta um Grid 2D onde consta os valores de (x, y) para que seja retornado o valor da substância.

Apesar do agente conter informações de três coordenadas, sua movimentação é realizada obedecendo à regra da vizinhança de *Moore*. Portanto, caso o agente tenha o objetivo de chegar a um determinado ponto, ele deve andar uma posição na sua vizinhança, não podendo pular posições. As direções a serem assumidas pelo agente estão ilustradas na Figura 3.3.

O parâmetro z do agente diz respeito à zona em que ele se encontra. Como neste modelo até o momento possui 5 zonas T na expressão abaixo pode assumir valores entre 1 e 5. A posição de um agente pode ser definida como:

$$P(i, j, z) = \{i \in N : 0 \leq i < W, j \in N : 0 \leq j < H, z \in N : 1 \leq z <= T\}$$

Como o espaço é finito, para evitar que um agente mova para fora das bordas da projeção (borda superior, inferior, direita e esquerda) foi implementado um procedimento que impede esta ação. Quando um agente estiver por exemplo na posição P(W -1, j, 1) e ao tentar mover para a posição P(W, j, 1), o *framework*

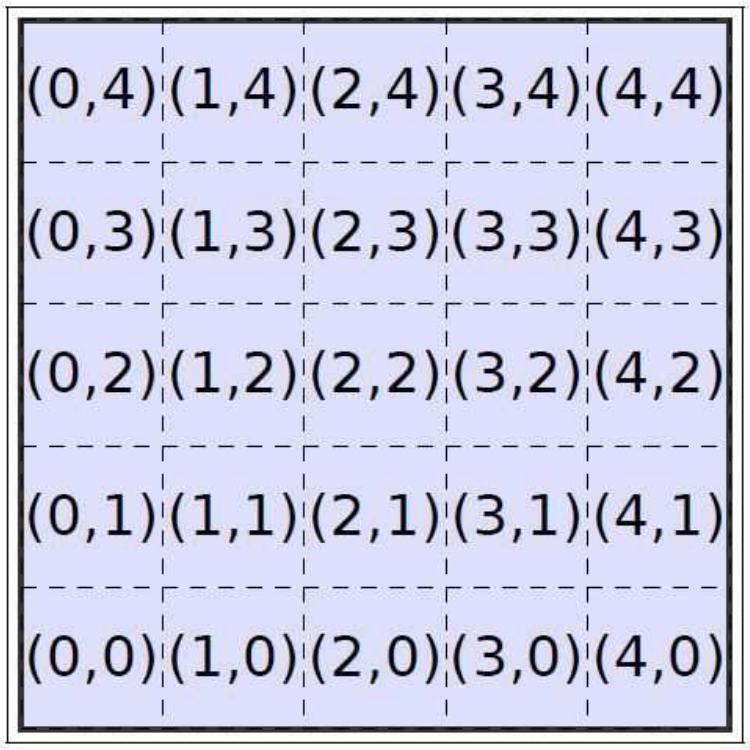

Figura 3.2. Ilustração da estrutura do espaço definido por uma projeção 2D.

Fonte: [Possi, 2012]

pode escolher duas alternativas, dependendo sorteio. A primeira é migrar o agente para a posição P(0, j, 1). Outra opção é não mover o agente. A movimentação dos agentes em linha é similar à coluna. É possível que dois agentes ocupem a mesma posição. Segundo Possi [2012], este comportamento simula um espaço 3D.

O fato dos agentes ficarem contornando o espaço dá a impressão que o mesmo é maior do ponto de vista do agente. Nesta situação, diz-se que o agente está em um espaço toroidal [Possi, 2012]. A Figura 3.4 ilustra uma representação deste espaço.

3.6 Parâmetros

Os parâmetros necessários para executar o modelo estão agrupados em uma biblioteca ***global.h***.

Todo agente que necessite acessar os parâmetros do modelo faz referência a biblioteca no seu cabeçalho. Basta adicionar a diretiva ***#include "../global.h"***. A Tabela 3.1 apresenta a relação de todos os parâmetros utilizados no modelo.

Todos os parâmetros foram criados de acordo com o padrão utilizado por Possi [2012]. Porém alguns parâmetros foram eliminados, como exemplo a dimensão

Tabela 3.1. Parâmetros do modelo

Parâmetro	Tipo	Valor Padrão
CONSTANT_REDUCTION	FLOAT	0,92
CONSTANT_DIFFUSION	FLOAT	0,5
CONSTANT_NUMBER_LAYERS_TISSUE	INTEGER	5
CONSTANT_NUMBER_LAYERS_LYMPHNODE	INTEGER	5
AFFINITYTHRESHOLD	INTEGER	5
TISSUEHEIGHTWIDTH	INTEGER	128
TISSUEQUARTER	INTEGER	8
PCPATTERN	STRING	01000100
RECEPTOR_PATTERN_LENGHT	INTEGER	5
DIRECTION_PORTAL_IN	INTEGER	0
DIRECTION_PORTAL_OUT	INTEGER	1
LINPHNODEHEIGHTWIDTH	INTEGER	50
CIRCULATIONHEIGHTWIDTH	INTEGER	256
PORTALPK1THRESHOLD	FLOAT	1,00
PORTAL_MK1_THRESHOLD	FLOAT	1,00
PORTAL_MACROPHAGE_TO_SEND	INTEGER	3
PORTAL_NK_TO_SEND	INTEGER	3
PORTAL_STATE_FUNCIONAL	INTEGER	0
PORTAL_STATE_NONFUNCIONAL	INTEGER	1
STRESSEDMAXTIME	INTEGER	25
PCSTRESSTHRESHOLD	FLOAT	5,00
TISSUESTATE_NORMAL	INTEGER	0
TISSUESTATE_STRESSED	INTEGER	1
TISSUESTATE_NECROSIS	INTEGER	2
TISSUESTATE_APOPTOSIS	INTEGER	3
TISSUELAYERCITOKINE_PK1	INTEGER	0
TISSUELAYERCITOKINE_MK1	INTEGER	1
TISSUELAYERCITOKINE_CK1	INTEGER	2
TISSUELAYERCITOKINE_APOPTOSIS	INTEGER	3
TISSUELAYERCITOKINE_NECROTIC	INTEGER	4
LYMPHNODELAYERCITOKINE_PK1	INTEGER	0
LYMPHNODELAYERCITOKINE_MK1	INTEGER	1
LYMPHNODELAYERCITOKINE_CK1	INTEGER	2
LYMPHNODELAYERCITOKINE_APOPTOSIS	INTEGER	3
LYMPHNODELAYERCITOKINE_NECROTIC	INTEGER	4
VIRUSINITIAL	INTEGER	100
VIRUSLATENCY	INTEGER	10
VIRUSVIRULENCY	INTEGER	2
VIRUSSELFPATTERN	STRING	00111100
TARGETPATTERN	STRING	01000100
TISSUE_ZONE	INTEGER	1
LYMPHMODE_ZONE	INTEGER	2
CIRCULATION_ZONE	INTEGER	3
BONE_MORROW_ZONE	INTEGER	4
THYMUS_ZONE	INTEGER	5
MACROPHAGEPATTERN	STRING	00010001
MACROPHAGELIFETIME	INTEGER	20
MACROPHAGEMK1DURATION	INTEGER	20
DENDRITIC_LYMPHNODE_LIFETIME	INTEGER	200
DENDRITIC_PK1_ACTIVATION_THRESHOLD	FLOAT	0,50
DENDRITIC_MK1_ACTIVATION_THRESHOLD	FLOAT	0,50
DENDRITIC_TISSUE_MK1_DURATION	INTEGER	30
DENDRITIC_LYMPHNODE_MK1_DURATION	INTEGER	20
DENDRITICSELFPATTERN	STRING	01010101
DENDRITICCELLSCOUNT	INTEGER	200
CTL_PROLIFERATION_COUNT	INTEGER	40
CTL_SELF_PATTERN	STRING	01110111
CTL_TISSUE_LIFETIME	INTEGER	300
CTL_LYMPHNODE_LIFETIME	INTEGER	500
CTL_LYMPHNODE_CK1_DURATION	INTEGER	100
CTL_CK1_MEMORY_THRESHOLD	FLOAT	0,20
CTL_TISSUE_CK1_DURATION	INTEGER	50
CTL_PK1_MEMORY_THRESHOLD	FLOAT	0,20
CTL_STATE_APOPTOSIS	INTEGER	0
CTL_STATE_INACTIVE	INTEGER	1
CTL_STATE_ACTIVE	INTEGER	2
CTL_STATE_MEMORY	INTEGER	3
BCELL_MK1_MEMORY_THRESHOLD	FLOAT	0,02
BCELL_CK1_MEMORY_THRESHOLD	FLOAT	0,02
BCELL_GERMINAL_PROBABILITY	INTEGER	60
BCELL_SELF_PATTERN	STRING	01100110
BCELL_LIFETIME	INTEGER	400
BCELL_ANTIBODY_COUNT	INTEGER	40
BCELL_STATE_INACTIVE	INTEGER	0
BCELL_STATE_WAITING	INTEGER	1
BCELL_STATE_GERMINAL	INTEGER	2
BCELL_STATE_PLASMA_CELL	INTEGER	3
BCELL_STATE_MEMORY	INTEGER	4
BCELL_STATE_APOPTOSE	INTEGER	5
TH_MEMORY_PROLIFERATION_COUNT	INTEGER	20
TH_CK1_MEMORY_THRESHOLD	FLOAT	1,00
TH_CK1_DURATION	INTEGER	50
TH_SELF_PATTERN	STRING	10001000
TH_LIFETIME	INTEGER	500
THCELL_STATE_INACTIVE	INTEGER	0
THCELL_STATE_ACTIVE	INTEGER	1
THCELL_STATE_APOPTOSIS	INTEGER	2
THCELL_STATE_MEMORY	INTEGER	3
NK_NOKILL_TIMEOUT	INTEGER	20
NK_CK1_DURATION	INTEGER	10
NK_SELF_PATTERN	STRING	00110011
NK_KILL_LIMIT	INTEGER	2
NK_LIFETIME	INTEGER	30
LYMPHNODE_WIDTH	INTEGER	56
LYMPHNODE_HEIGHT	INTEGER	56

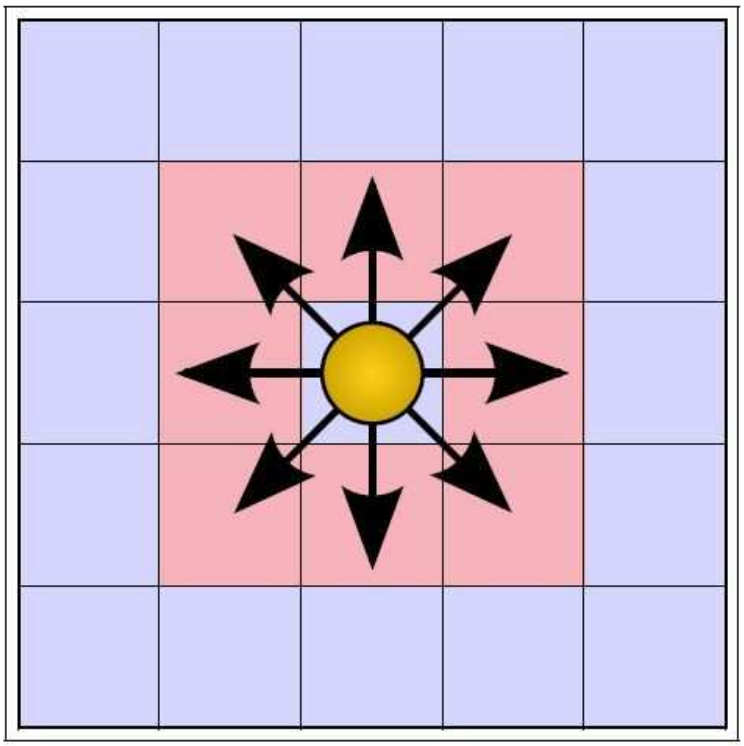

Figura 3.3. Vizinhança de Moore.

Fonte: [Possi, 2012]

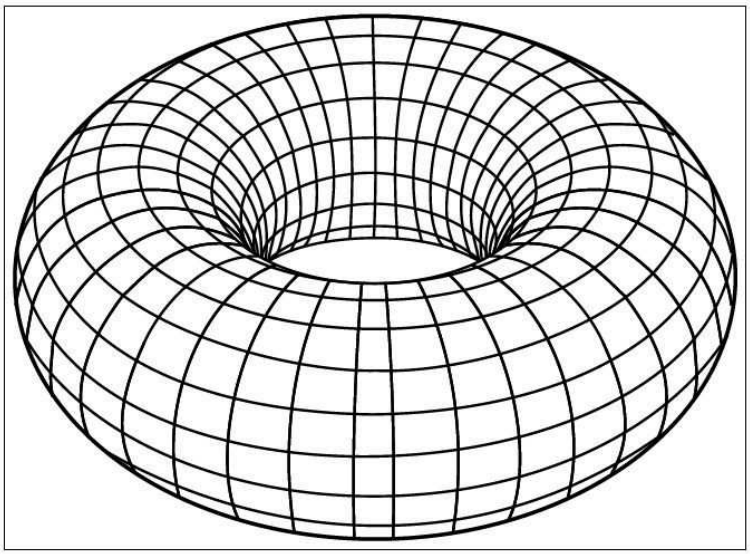

Figura 3.4. Ilustração de um espaço toroidal.

Fonte: [Possi, 2012]

de algumas zonas, neste Possi [2012] utilizava dois parâmetros para especificar a dimensão da zona *TISSUE*, ficando apenas *TISSUEHEIGHTWIDTH*.

3.7 Zonas

Como nos trabalhos de [Possi, 2012; Folcik et al., 2007], este trabalho também possui 5 zonas. A implementação difere dos autores citados, total ou parcialmente, mas a função é mantida.

O Flame não apresenta uma estrutura de dados pré-definida para indicar ZONA, mas flexibiliza a implementação das mesmas. Cada zona tem sua camada de dados (*grid*) que serve para armazenar os valores de substâncias. Os *grids* estão acoplados ao ambiente, conforme citado anteriormente.

As zonas representam a área do organismo onde as células realizam suas atividades. Existem vários agentes que são criados nestas zonas e outros que estão migrando de região passando por elas. As substâncias por sua vez também circulam nas zonas.

As zonas presentes neste modelo são: *Tissue, Lymphnode, Circulation, Bone-Marrow e Thymus*. É importante ressaltar que o parâmetro z indica a zona em que o agente se encontra. Segue a descrição de cada zona:

3.7.1 Tissue

De acordo com Folcik et al. [2007], esta zona reproduz uma fatia microscópica de um tecido parenquimatoso genérico e é nesta que ocorrem os primeiros contatos das células do organismo com patógenos e aspecto típico das infecções. Esta é a principal zona do modelo.

As principais células desta zona são as células parenquimatosas, representadas neste modelo pelos agentes PC. Os agentes são organizados em quarteirões, formando o espaço da zona, que representa a organização do tecido como um órgão. Em alguns *framework*, como o usado por Possi [2012], é possível visualizar esta zona. Já o Flame não dispõe desta funcionalidade.

A formação dos quarteirões segue parâmetros como *TISSUEHEIGHTWIDTH* que define a área total do tecido. O parâmetro *TISSUEQUARTER* define o tamanho do quarteirão.

Uma correspondência à área real do tecido é feita por Possi [2012]. Este autor declara que caso o tamanho da zona *Tissue* seja 110 unidades, corresponderia a cerca de 1,21mm^2. Isto baseado em informações de Folcik et al. [2007] que considera em seu trabalho que uma célula possui 0,01mm. Os agentes que estão nesta zona, têm o valor da coordenada z igual ao parâmetro *TISSUE_ZONE*.

3.7.2 Lymphnode

Esta zona representa o linfonodo, mas de acordo com Folcik et al. [2007] ela também pode representar o baço. É nesta zona que ocorre o reconhecimento de antígenos pelos linfócitos, onde os antígenos são apresentados. Em caso de reconhecimento, eles se proliferam migrando para outras regiões.

Portanto, os principais agentes presentes neste local são os linfócitos T *helper*, os linfócitos T citolíticos (CTL) e os linfócitos B (*BCell*).

O tamanho desta zona é definido pelo parâmetro *LINPHNODE-HEIGHTWIDTH*. Foi definido com o valor 56 unidades, para manter compatibilidade com o trabalho de [Possi, 2012]. Agentes que estejam nesta zona têm a coordenada z igual ao parâmetro *LYMPHMODE_ZONE*.

3.7.3 Circulation

Esta zona reproduz o funcionamento da circulação linfática e sanguínea. Sua principal função é simular as viagens dos agentes (células) do *Lymphonode* para o *Tissue*. O tempo que uma célula leva para fazer esta viagem não é conhecido [Folcik et al., 2007].

A principal população de agentes desta zona são os linfócitos que vêm do *Lymphnode* com destino a *Tissue*. As dimensões desta zona é definida pelo parâmetro *CIRCULATIONHEIGHTWIDTH*, que teve seu valor definido como 256 unidades. Já os agentes que circulam nesta zona têm o valor da coordenada z igual ao parâmetro *CIRCULATION_ZONE*.

3.7.4 Bone Marrow

Esta zona retrata o funcionamento da medula óssea. Tem como principal função efetuar os mecanismos relacionados à tolerância central das células B, já que os agentes possuem uma representação de especificidade.

Além disto, esta zona gera os linfócitos B com especificidade aleatória e faz os testes com os mecanismos da tolerância central do linfócito B. Agentes que estejam nesta zona têm o valor da coordenada z igual ao parâmetro *BONE_MORROW_ZONE*.

Tolerância Central do Linfócito B: A Figura 3.5 apresenta o funcionamento da tolerância central. O linfócito B é gerado nesta zona e recebe o nome de pré-B, sem especificidade. Posteriormente, este linfócito recebe uma especificidade aleatória e é submetido à geração de diversidade, passando a ser um linfócito imaturo.

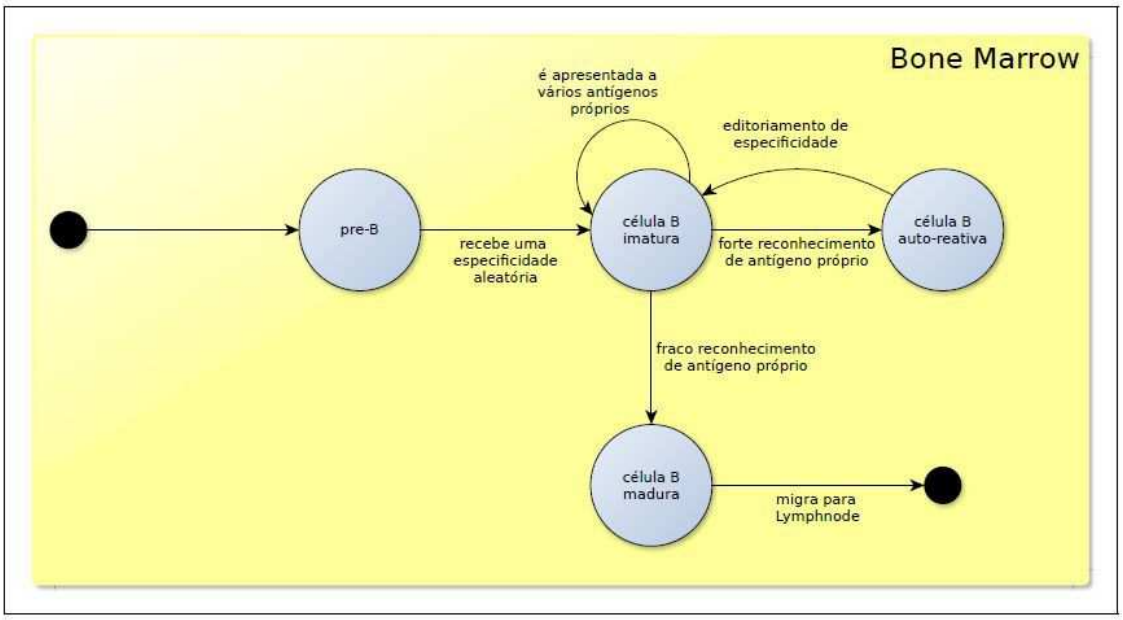

Figura 3.5. Tolerância Central do Linfócito B.

Fonte: [Possi, 2012]

Então, o linfócito é exposto a um banco de dados de antígenos próprios. Em caso de reconhecimento de autoantígenos com alta afinidade, o linfócito tem seus receptores alterados e volta a ser um linfócito B imaturo. Quando o reconhecimento é de baixa afinidade, acontece a seleção negativa e o mesmo segue para os órgãos linfoides [Abbas et al., 2012].

3.7.5 Thymus

Esta zona representa o Timo e tem como principal função a maturação das células T geradas na medula óssea, assim como os linfócitos T *helper* (*ThCell*) e T citolítico (CTL). Eles são gerados com especificidade aleatória e posteriormente são submetidos ao mecanismo de tolerância central do linfócito T. Agentes contidos nesta zona têm o valor da coordena z igual ao parâmetro *THYMUS_ZONE*.

Tolerância Central do Linfócito T: Os linfócitos T *helper* e T citolítico são gerados sem especificidade e recebem o nome de pré-T. Posteriormente, estes recebem dois receptores TCR (CD4+ e CD8+) e mudam para linfócito T duplo-positivo.

A próxima etapa é apresentar autoantígenos presentes no banco de dados para estes linfócitos. O reconhecimento de algum autoantígeno com alta afinidade, causa a eliminação do linfócito. No entanto, o reconhecimento do autoantígeno com baixa afinidade pelo MHC-I transforma o linfócito em um linfócito T citolítico. O recon-

hecimento pelo MHC-II com baixa afinidade faz com que o linfócito vire um linfócito T *helper*. Caso o linfócito vença esta etapa, ele migra para o *Lymphnode*. A Figura 3.6 ilustra o procedimento descrito acima. Esta figura apresenta o comportamento do linfócito no *Thymus*, uma vez que o mesmo foi gerado na medula óssea

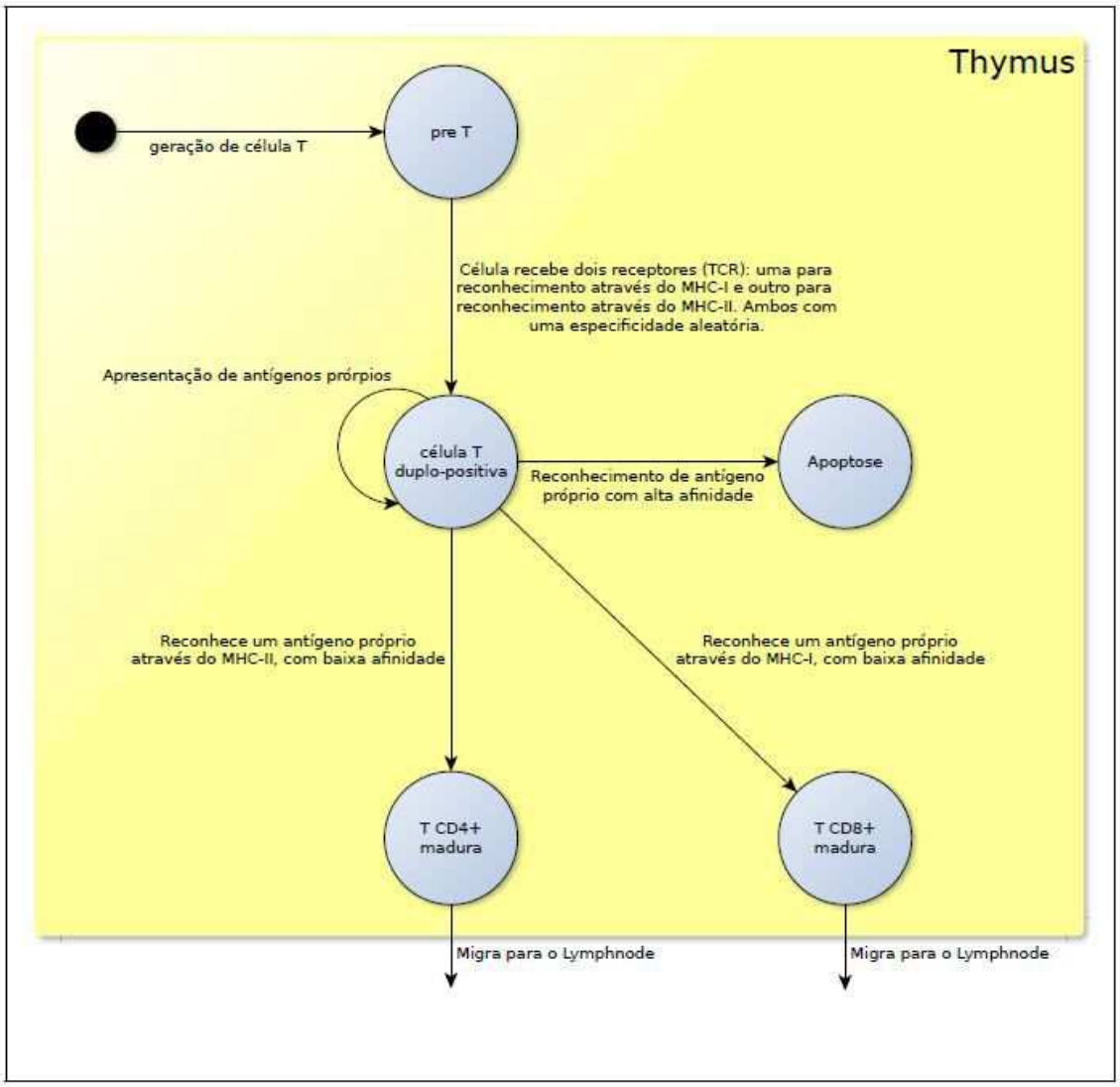

Figura 3.6. Tolerância Central do Linfócito T.

Fonte: [Possi, 2012]

3.7.6 Migração entre zonas

Cada agente é gerado em uma zona específica e ao decorrer de algum tempo migra para outra. Como exemplo, os linfócitos criados na zona *BoneMarrow* e *Thymus*, migram para a zona *Lymphonode*. Já os linfócitos ativos do *Lymphnode* migram para

a *Circulation* e posteriormente para a *Tissue*. As células dendríticas (*Dendritics*), quando capturam antígenos em posições que representam *danger signal*, tendem a migrar para o *Lymphone*, onde apresenta o antígeno aos linfócitos presentes nesta zona.

As migrações são realizadas por agentes específicos chamados portais. De acordo com Folcik et al. [2007], estes agentes simulam o funcionamento dos vasos linfáticos que fazem o transporte de células e substâncias entres as zonas.

Os portais ficam posicionados em pontos fixos nas zonas. Quando um agente alcança-os e atende aos requisitos, o portal transfere o agente para outra zona. Isto ao alterar a coordenada z do agente.

Então, quando um agente atinge o *circulationportal*, pode ser levado para a zona *Tissue*. Este procedimento simula a passagem de um agente por um vaso sanguíneo. O mesmo acontece com substâncias, uma vez que o valor da concentração da substância em uma extremidade do portal é liberada na outra extremidade.

3.8 Difusão de substâncias

As citocinas ou mediadores químicos, quando liberados, desempenham duas atividades fundamentais que são: a difusão que consiste no espalhamento pela vizinhança do local onde foi liberada, diminuindo sua concentração e o decaimento, que é a diminuição do valor liberado ao passar do tempo. Este procedimento é fundamental, uma vez que as células usam as citocinas para se comunicarem.

Este processo ocorre nas camadas de dados que estão no ambiente, e para realizar a difusão, os agentes devem usar os métodos:

- para liberar substâncias na pele utiliza-se o método **void releaseCitokine(int x, int y, int zoneCitokine)**. A coordenada (z, y) é o local onde a substância será liberada, o valor é obtido pelo parâmetro **CITOKINEVALUE**. Já o *zoneCitokine* é a camada de dados usada na *Tissue* referente à substância que se deseja liberar. Estas substâncias serão descritas posteriormente. Porém seus valores podem ser obtidos através dos parâmetros (**TISSUELAYERCITOKINE_PK1, TISSUELAYERCITOKINE_MK1, TISSUELAYERCITOKINE_CK1, TISSUELAYERCITOKINE_APOPTOSIS, TISSUELAYERCITOKINE_NECROTIC**).
- quando o agente precisa saber o valor de uma substância em determinada posição, o mesmo deve utilizar o método: **float getCitokineValue(int x,**

int y, int zoneCitokine). As coordenadas (x, y) são o local cartesiano da concentração. O *zoneCitokine* é a camada de dados. Os valores são os mesmos citados acima.

A estrutura de dados foi dividida por zonas, ou seja, os métodos citados acima são da *Tissue*; já o *Lymphnode* possui seus próprios que são: **void releaseCitokine-Lymphnode (int x, int y, int zoneCitokine)**, para Liberar substâncias; e **float getCitokineValueLymphnode(int x, int y, int zoneCitokineLymphnode)**, obtém o valor da substâncias. No parâmetro zoneCitokine aceita os seguintes valores: (**LYMPHNODELAYERCITOKINE_PK1, LYMPHNODELAY-ERCITOKINE_MK1, LYMPHNODELAYERCITOKINE_CK1, LYM-PHNODELAYERCITOKINE_APOPTOSIS, LYMPHNODELAYERCI-TOKINE_NECROTIC**).

Os métodos descritos acima apenas liberam ou obtêm o valor da substância. Porém, a substância deve passar pelo processo de difusão. Neste modelo, o método que realiza esta atividade é o **int diffusion_matrix()**.

Este método envia para GPU as camadas de dados para executar o cálculo da equação 3.1. O procedimento trabalha com duas matrizes: atual e auxiliar. Abaixo é apresentada a equação para realizar a difusão de substâncias.

$$V\,novo = E * (Vatual + D * (\overline{Vvizinhos} - Vatual)) \tag{3.1}$$

Onde:

E: constante de caimento;

D: constante de difusão;

Vnovo: Valor da concentração atualizado na posição atual;

Vatual: valor da concentração antigo (do *tick* anterior) na posição atual;

Vvizinhos: Valor da concentração antigo (do *tick* anterior) na vizinhaça de *Moore* [Possi, 2012].

As duas matrizes são linearizadas e enviadas para GPU, que faz o cálculo em todas as posições. No entanto, para agilizar o processo de envio e recebimento de dados entre CPU e GPU, apenas o conjunto de informação atualizado é retornado. Então, a camada de dados tem suas duas matrizes atualizadas, ou seja, neste ponto tanto a matriz real como a auxiliar estão com os mesmos valores. Assim que os agentes executam suas tarefas e a difusão é realizada, o próximo *tick* é iniciado.

A Figura 3.7 mostra como são atualizados os valores nas camadas de dados, através da difusão de substâncias. Em 3.7 (a) é apresentado o gradiente contínuo, ou seja, como é o mundo real. Foi realizada uma liberação de substância na célula que tem o valor 10.

Já na Figura 3.7(b) é ilustrada a discretização do gradiente, ou seja, como os agentes visualizam a difusão de substâncias.

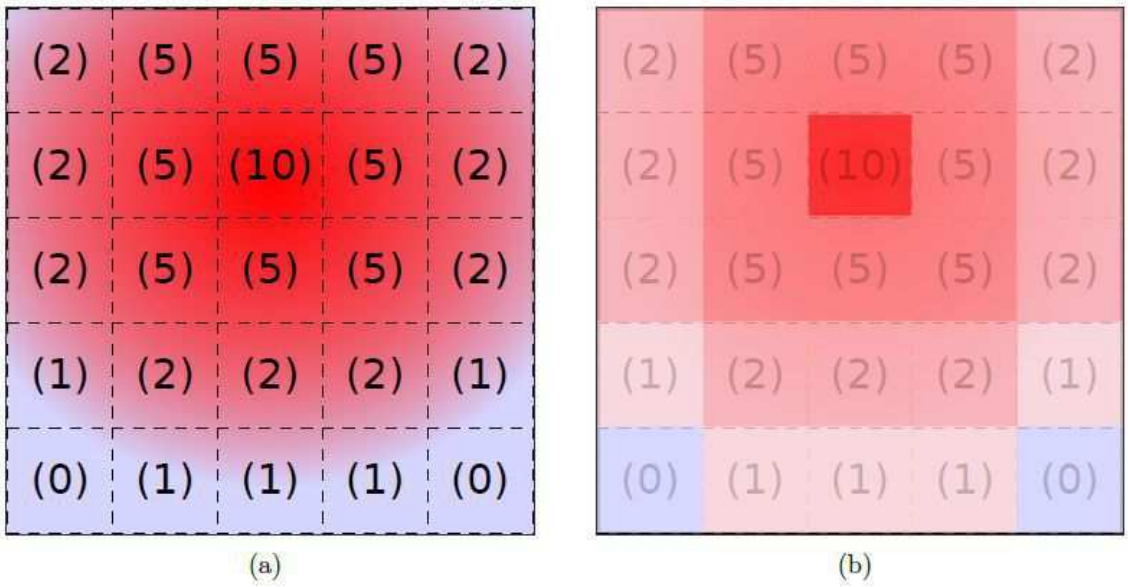

Figura 3.7. Ilustração da representação do gradiente de dispersão das citocinas através das camadas de dados. Em (a) distribuição contínua da substância, como acontece no mundo real. Em (b), discretização do gradiente, como os agentes veem

Fonte: [Possi, 2012]

3.8.1 Citocinas

As substâncias presentes neste modelo retratam as presentes no mundo real. Porém se baseia em um sub-conjunto das substâncias apresentadas por Folcik et al. [2007].

Segundo Folcik et al. [2007], para representar substâncias reais em um modelo de simulação é necessário agrupá-las por funcionalidade. Este mesmo autor agrupa as substâncias em: substâncias que estendem a resposta inflamatória que são as pró-inflamatórias e substâncias que reduzem a resposta inflamatória chamadas de anti-inflamatórias.

Nos trabalhos de [Folcik et al., 2007; Possi, 2012], foi utilizado um padrão para identificar o grupo a que a substância pertence. Se o nome da substância terminar

com o número 1, então é uma pró-inflamatória. No entanto, se terminar com o número 2 é anti-inflamatória. Neste trabalho, será utilizada esta nomenclatura.

Abaixo serão descritas as substâncias presentes neste modelo e suas representações reais, baseado em [Folcik et al., 2007; Possi, 2012].

As substâncias pro-inflamatórias são:

- **PK1** é baseada na *"Parenchymal-kine 1"*, apresentada por Folcik et al. [2007]. Tem como função representar situações de estresse causadas por danos sofridos pelo tecido, ou pela resposta imunológica. Este mesmo autor declara que fatores de estresse como choque-térmico (HSP) são apresentados por esta substância, também ácido úrico e etc. Além das quimiocinas como CX3CL1, CCL3, CCL5, e CCL6.

- **MK1** assemelha com a *"Mono-kine 1"* (MK1), apresentado por Folcik et al. [2007], e reproduz o conjunto de substâncias pró-inflamatórias da resposta imunológica inata. De acordo com o mesmo autor, esta corresponde às substâncias IL-12, IL-8, CCL3, CCL4, CCL5, CXCL9, CXCL10 e CXCL11.

- **CK1** é baseada na *"Cytokine 1"*, apresentada por Folcik et al. [2007]. Corresponde às substâncias pró-inflamatórias presentes na resposta imunológica adaptativa. Ainda, segundo este autor, estas substâncias são correspondentes a INF-γ, IL-2 e TNF-β.

- **NECROSIS** é baseada na *"Necrosis factors"*, apresentada por Folcik et al. [2007]. Ainda, segundo este autor, esta substância corresponde a restos mortais de células que foram mortas por processo traumático, e não por morte programada.

As substância anti-inflamatórias presentes neste modelo são:

- **MK2** assemelha com a substância *"Mono-kine 2"* (MK2), apresentada por Folcik et al. [2007]. Corresponde a substâncias anti-inflamatórias existentes na resposta imunológica inata. De acordo com o mesmo autor, esta substância representa IL-10, CCL1, CCL17, CCL22, CCL11, CCL24, CCL26.

- **CK2** é baseada na substância *"Cytokine 2"* (CK2), apresentada por Folcik et al. [2007]. Corresponde às substâncias anti-inflamatórias existentes na resposta imunológica adaptativa. De acordo com o mesmo autor esta substância representa TGF-β, IL-4, IL-5, IL-6, IL-10, IL-13 e INF-γ.

- **APOPTOSIS** é baseada na substância *"Apoptotic bodies"*, apresentada por Folcik et al. [2007]. Corresponde aos restos mortais de células que sofreram morte celular programada ou apoptose.

Assim como Possi [2012], não incluiu todas as substâncias presentes em Folcik et al. [2007], pois a granulidade utilizada possibilitou que algumas substâncias fossem modeladas como agentes. Este trabalho segue a abordagem utilizada por Possi [2012].

3.9 Agentes

Serão apresentados os agentes presentes neste modelo e suas funções. Como o *framework* Flame utiliza a linguagem de programação C, os agentes foram descritos como arquivos ".c".

3.9.1 Granularidade

A granularidade está relacionada ao nível de abstração que é adotado no modelo. Pois quanto mais fina a granularidade, mais detalhes terá e mais complexo será o modelo. Em contra partida, quanto mais grossa a granularidade, mais detalhes terão de ser abstraídos, podendo em alguns casos deixar alguma funcionalidade importante fora do modelo [Possi, 2012].

O ideal é que se modele utilizando a granularidade mais fina, porém esta demanda um alto custo computacional. Sendo assim, a utilização da granularidade mais grossa é utilizada, desde que atenda ao comportamento desejado [Folcik et al., 2007]. Como o propósito é modelar o SI, funcionalidades como especificidade, afinidade e tolerância devem estar presentes.

Neste trabalho, o nível de granularidade utilizada é *antigen-as-agent* ou *hasEspecificity-as-agent*. Portanto, antígenos ou qualquer outro objeto que contenha antígeno será modelado como agente. Dado que, qualquer componente reconhecido por PRR é considerado agente. As sustâncias como citosinas e outras moléculas, estão representadas pelas camadas de dados. Deste modo, este modelo se assemelha com o apresentado por [Possi, 2012].

A Figura 3.8 mostra os elementos modelados como agentes, assim como as caixas que são utilizadas para troca de mensagens. Nesta figura, é possível perceber o nível de granularidade escolhido.

3.9.2 Especificidade

Este termo está relacionado à resposta específica (resposta imunológica adaptativa) para um determinado invasor, ou seja, é quando o linfócito é produzido especificamente para combater um determinado antígeno. Isto é realizado através de um padrão expresso pelo agente, que indica o alvo de suas ações [Possi, 2012]. Segundo o mesmo autor, a especificidade é dividida em dois conceitos (antígenos e especificidade):

- **Antígenos**: é o padrão identificador de qualquer agente que apresente antígenos. Pois, existem agentes no modelo que não apresentam antígenos (os portais).

- **Especificidade**: Segundo [Possi, 2012], *"é o padrão dos receptores dos linfócitos e dos anticorpos, o padrão para o qual eles são específicos, ou o padrão que os ativa"*. Fazendo uma analogia com o mundo real, o padrão é chamado de padrão molecular e tem como estrutura uma sequência de aminoácidos.

  No entanto esta representação no modelo será feita por uma sequência de bits 0 ou 1, também chamada de *string* de *bits* [Floreano & Mattiussi, 2008].

  Exemplo: [01010101] ou [00110011]

  No Flame os atributos dos agentes são modelados em um arquivo XML. Porém todo agente que possui antígeno contém um atributo similar a **<pcSelfPattern>00000000</pcSelfPattern>**. Neste exemplo é apresentado o padrão da célula PC. O valor do padrão é alterado no momento da criação do agente.

  No entanto, os linfócitos possuem um padrão para representar sua especificidade, também chamado de *target*. Exemplo: o agente CTL **<Ctltarget>00000000</Ctltarget>**. O valor é alterado no momento da criação do agente.

- **Antígenos Próprios**: os agentes que passam pela tolerância central, possuem um banco de dados de antígenos próprios, também chamado de conjunto SELF. Este conjunto é apresentado ao linfócito no processo de tolerância central. Neste modelo os agentes são: CTL, PC, APC, NK, *ThCell*, *BCell*, *Macrophage*, *Dendritic*.

3.9.3 Afinidade

O termo afinidade refere-se ao reconhecimento ou similaridade entre os agentes e é verificada ao confrontar os padrões dos agentes. Ou seja, ao comparar o padrão de um antígeno com o seu receptor é possível verificar o grau de afinidade entre eles.

Floreano & Mattiussi [2008], apresenta um método para calcular o grau de afinidade entre dois padrões de *bits*, chamado de *"longest commom substring lenght"*, que Possi [2012] traduziu como (comprimento da maior sequência comum). Este procedimento trata-se de: dado duas sequencias de *bits* A e B, encontrar a maior sequência de caracteres (*bits*) contíguos em A e B simultaneamente na mesma ordem.

Logo o grau de afinidade é expressado como o comprimento da maior sequência entre os padrões analisados (A, B). Seguem alguns parâmetros que o cálculo utiliza:

AFFINITYTHRESHOLD: tamanho do comprimento mínimo que indica se os dois padrões se reconhecem.

RECEPTOR_PATTERN_LENGHT: comprimento padrão da sequência de bits que representa a especificidade do receptor.

O método **"int match(char* caracter1, char* caracter2)"** é responsável por fazer o cálculo da afinidade e foi implementado no agente que representa o ambiente. Uma característica deste método é que ele faz a verificação com *string* de *bits* de tamanhos variados. A Figura 3.9 ilustra a execução deste método.

Os agentes (CTL, PC, APC, NK, *ThCell*, *BCell*, *Macrophage*, *Dendritic*), possuem MHCI e capacidade de apresentar o padrão SELF. Quando infectado, o padrão do patógeno que infectou, representa o complexo de histocompatibilidade principal I.

Os agentes APC (*BCell, Macrophage, Dendritic*), possuem o MHCII capaz de apresentar o padrão *SELF* de agentes que foram fagocitado. Portando representa o complexo de histocompatibilidade principal II.

Já os linfócitos CTL, *ThCell*, *BCell*, possuem receptores e uma especificidade. Estes utilizam o cálculo de afinidade a fim de identificar se seus receptores reconhecem antígenos apresentados pela APC.

Os agentes *Macrophage* e NK usam o cálculo de afinidade para identificar substâncias como PAMPs e para verificar o MHCI das células. Isto na imunidade inata. Segundo Possi [2012] PAMPs refere-se ao conjunto de padrões que foram selecionados e armazenados em banco de dados de antígenos de PAMPs.

3.9.4 Tipos de Agentes

A Figura 3.8 ilustra os vários tipos de agentes criados para atender ao modelo, juntamente com o nível de granularidade escolhido. Os agentes são do tipo reativo com estado, pois esta característica assemelha-se ao comportamento real das células do organismo. Será descrito nas subseções os agentes do modelo e suas principais funções. No entanto, na seção a seguir, serão apresentadas características comuns a todos os agentes.

3.9.4.1 Informações gerais de agentes

Como citado anteriormente, o Flame realiza a execução dos agentes obedecendo uma ordem pré-estabelecida no arquivo XML. Caso necessite que algum procedimento seja executado em um determinado *tick*, é necessário fazer o controle manualmente. Para auxiliar este procedimento, o *framework* disponibiliza a variável ***"iteration_loop"*** que retorna o número do *tick* em execução.

Os principais atributos dos agentes são:

- **x** tipo Integer que armazena o valor da coordenada x, e representa a posição do agente no espaço;
- **y** tipo Integer que armazena o valor da coordenada y, e representa a posição do agente no espaço;
- **z** tipo Integer que armazena o valor da coordenada z, e representa a zona em que o agente se encontra;
- **age** indica a idade do agente. Em algumas situações é necessário controlar quanto tempo o agente vai participar da simulação;
- **SelfPattern** representa o padrão molecular do agente. Atributos deste tipo possuem as iniciais do agente. Exemplo: *pcSelfPattern*. Este procedimento elimina o agente *Antigen* proposto no modelo de [Possi, 2012].

Todos os métodos que são utilizados por mais de um agente foi implementado no ambiente. Isto para evitar duplicidade de código. Portanto, os agente usam os métodos do ambiente.

Os estados pelos quais os agentes passam no decorrer da simulação foram modelados com um atributo ***"currentstate"***. Portanto, não foi criado método

para alterar o estado do agente. Exemplos dos possíveis valores que o agente ***Th-Cell*** pode assumir *(THCELL_STATE_INACTIVE, THCELL_STATE_ACTIVE, THCELL_STATE_APOPTOSIS, THCELL_STATE_MEMORY.*

No entanto, um agente não acessa o outro diretamente. Para alterar o estado do outro, basta adicionar esta informação em uma caixa de mensagem que o agente verifica e executa a ação. Este procedimento é ilustrado na Figura 3.8.

3.9.4.2 Vírus

Este agente é classificado como um patógeno e sua tarefa é infectar as células do organismo. Possui um atributo que representa o padrão expresso pelo vírus ***"virus-SelfPattern"***. Contudo, este agente possui um parâmetro que indica qual o padrão que o mesmo tem afinidade para atacar *target*, ou seja, o alvo. No modelo, este parâmetro é nomeado como ***"virusTargetPattern"***.

Este agente imita o comportamento de um vírus real, pois circula pela *Tissue*. Ao encontrar uma célula hospedeira que possui um padrão reconhecido pelo *target* do vírus, tenta infectá-la. Caso a célula não esteja infectada. Segundo Possi [2012], infectar a mesma célula duas vezes consiste em desperdício de agentes.

Após a infecção, o vírus insere seu padrão na célula que é apresentado pelo MHCI. Ao passar algum tempo a mesma começa a criar réplicas do vírus, inclusive com o padrão antigênico da célula. De acordo com Possi [2012], existe a possibilidade do vírus modificar seu padrão e se misturar com as células.

Assim como o modelo do autor referido não contempla este recurso, este modelo também não. A Figura 3.10 ilustra o comportamento descrito acima.

3.9.4.3 Portal

Os portais foram modelados como agentes e além de servirem para realizar a migração de agentes e substâncias entre zonas, segundo Folcik et al. [2007], eles também representam os vasos sanguíneos, linfáticos e a circulação.

- **TissuePortal:** estes representam os vasos sanguíneos e linfáticos que irrigam o tecido e são conectados aos portais que estão na zona *Circulation*. Quando são detectados sinais de estresse das células teciduais (PK1), este agente cria *Macrophage* e NK [Folcik et al., 2007].

- **Circulation Portal:** estes agente representam os vasos sanguíneos e linfáticos que atuam na captura das células na circulação e as levam para o sítio da infecção. No entanto, alguns estão conectados a *Tissue* e levam agente para

esta zona. Outros estão conectados à zona *Lymphnode* e tem a função de trazer os linfócitos desta zona.

- **LymphnodePortal:** este tem a função de irrigar os linfonodos e o baço [Folcik et al., 2007]. Também estão conectados à zona *Circulation*, para onde levam os linfócitos ativos.

3.9.4.4 Parenchyma Cell

Estes agentes foram implementados no arquivo PC.c e representam o tecido parenquimatoso de um órgão abstrato. Tem a função de representar a pele em uma estrutura de grade. Foi baseado no agente PC apresentado por Folcik et al. [2007]; Possi [2012].

No primeiro *tick*, o próprio agente, ou seja o primeiro agente cria uma estrutura que representa células de um órgão abstrato. Estas células são atacadas por patógenos que quando as invadem conseguem se reproduzir.

O comportamento deste agente é ilustrado na Figura 3.11. Estes agentes, quando no estado NORMAL, verificam se ocorreu algum problema com seu vizinho. Caso um vizinho tenha sido danificado e sofrido apoptose, este agente cria um outro para manter a estrutura. Este procedimento simula a regeneração do tecido.

Na situação em que uma célula seja lisada e passe a apresentar *NECROSIS*, a célula vizinha detecta esta substância e passa para o estado de *ESTRESSE*, onde libera PK1. Quando a substância *NECROSIS* for dissipada e ficar abaixo do limite, a célula que estava no estado de ESTRESSE volta ao estado NORMAL.

Quando um Vírus infecta uma célula, esta tem seu estado alterado e libera a substância PK1. Deste modo pode ocorrer duas situações: A primeira é a fagocitação que pode ser por: linfócito T citolítico (CTL), célula NK, *Macrophage* ou *Dendritic*. A segunda é quando o Vírus não é eliminado e supera seu período de latência definido pelo parâmetro ***VIRUSLATENCY***.

Quando o Vírus superou o período de latência, a célula é lisada e tem seu estado alterado para NECROSE. Logo libera novos vírus. O parâmetro ***VIRUSVIRULENCY*** define a quantidade de vírus que será liberado.

O agente PC tanto no estado de apoptose quanto necrose, continua na simulação para representar os restos mortais destas células. Mas em algum momento estes restos mortais serão eliminados pelos *Machophage*. Isto elimina o agente da simulação e permite que um novo agente seja criado (regeneração do tecido).

3.9.4.5 NK

O agente NK também chamado de *Natural Killer* tem como atividade principal a eliminação dos depósitos de infecção e células infectadas aptas a reproduzir vírus. Isto enquanto a imunidade adaptativa prepara sua resposta. De acordo com Folcik et al. [2007], a presença do complexo de histocompatibilidade classe I inibe a ação deste agente. Entretanto, em meios pró-inflamatórios esta ação é ignorada e o agente NK mata células estressadas.

A Figura 3.12 ilustra as funcionalidades deste agente. Ele é adicionado na simulação (na zona *Tissue*) quando o *TissuePortal* identifica substância pró-inflamatória MK1, indicando a quimiotaxia das células.

Este agente fica patrulhando o tecido e, ao perceber a substância PK1, vai em direção as células que estejam emitindo esta substância, pois as mesmas devem estar estressadas ou infectadas. Então, quando estas células são encontradas e existir grande concentração de MK1 no local, este agente induz a célula a apoptose. Logo, o agente é eliminado da simulação (sofre apoptose), quando seu tempo de vida esgota ou quando ele elimina uma quantidade de células.

3.9.4.6 ThCell

Sua principal atividade é fornecer estímulos para que o agentes *BCell* ou o *Machophage* seja ativado.

Este agente tem sua origem no *Thymus* e é submetido a algumas etapas de criação como: receber a especificidade e testar a tolerância central para impedir que o mesmo seja auto-reativo.

A Figura 3.13 ilustra a funcionalidade deste agente. Pois, após ser gerado no *Thymus* com status inativo, o agente migra para o *Lymphnodo* e fica a espera de um contato com um agente APC para ativá-lo. O procedimento de ativação (passar o *Thcell* para o status ativado) acontece quando um APC apresenta seus antígenos pelo MHCII e o Thcell os reconhece.

O *Thcell*, quando se encontra no estado ativo, libera a substância CK1 e reproduz gerando clones, que por sua vez serão enviados à camada *Tissue*. A quantidade de clones gerados é controlada pelo parâmetro **TH_MEMORY_PROLIFERATION_COUNT**. Caso o *ThCell* ativo no *Lymphonode* encontre um linfócito B, este será ativado.

Além de ativar um linfócito B como citado acima, o *Thcell* em caso de reconhecimento de um novo antígeno tem seu tempo de vida prolongado. Ao passar do tempo, caso o agente esteja em um local de baixa concentração de CK1, o mesmo

se torna uma célula de memória. Célula esta que responde rapidamente quando ativada por um agente APC.

No entanto, se o *ThCell* estiver em um local com alta concentração de CK1 e o seu tempo de vida esgotar, o mesmo é eliminado da simulação por meio da apoptose.

Na zona *Tissue* o *ThCell* segue as substâncias MK1 e CK1, em busca de focos de infecção. Caso este agente reconheça um antígeno apresentado por uma célula APC (no caso um *Machophage*), ele tem seu tempo de vida re-iniciado e continua a liberação de CK1. Por outro lado, caso este encontro não aconteça, o tempo de vida do *ThCell* se esgota, levando-o a apoptose.

3.9.4.7 CTL

Este agente representa os linfócitos T citolítico e é baseado no agente "CTL"definido por Possi [2012]. Segundo este autor, a principal função deste agente assim como o agente NK é eliminar os depósitos de infecção. No entanto, este agente se difere do NK, pois é necessário reconhecer o antígeno apresentado pelo MHCI para eliminar a célula infectada. De acordo com Possi [2012], o agente NK é capaz de descartar a apresentação do MHCI em situações pró-inflamatórias e fagocitar a célula infectada.

A Figura 3.14 apresenta o comportamento do agente CTL, que é concebido na zona *Thymus*, onde recebe sua especificidade e é testado para verificar se não é auto-reativo. Logo, este agente migra para a zona *Lynphnode*. Nesta etapa, o agente está no estado inativo e fica circulando à procura de uma célula APC. Ao encontrar esta célula, o CTL tenta reconhecer o antígeno que a mesma apresenta. Este procedimento é denominado apresentação cruzada (*cross-priming*) [Folcik et al., 2007; Abbas et al., 2012].

Em caso de sucesso no reconhecimento do antígeno, o CTL é ativado e prolifera, produzindo clones que migrarão para a *Tissue*. Este também passa a produzir a substância CK1.

Então este agente fica circulando pelo *Lymphnode* na tentativa de contatar outro agente APC. Caso ocorra, o procedimento anterior é repetido novamente, ou seja, o CTL gera os clones e produz mais CK1.

Na situação em que o agente fique o tempo definido pelo parâmetro ***CTL_LYMPHNODE_LIFETIME*** sem encontrar células APC e sem a presença de CK1, ele se torna uma célula de memória. Mas quando está na presença de CK1 e não encontra APC, o mesmo sofre apoptose.

Na zona *Tissue*, este agente se torna um efetor, segue PK1 e libera CK1. Elimina células que apresentam antígenos que o mesmo reconheça pelo MHCI. Em

cada reconhecimento, o tempo de vida é aumentado.

Quando não encontra substâncias PK1 e CK1, esta célula se torna de memória e anda livremente pelo tecido. Porém examina os antígenos apresentados pelo MHCI. Em caso de sucesso no reconhecimento, torna-se uma célula efetora. Portanto, caso o tempo de vida do agente se esgote, o mesmo é eliminado da simulação pelo processo de apoptose.

3.9.4.8 APC

Este trabalho é uma proposta de modelagem do SI e foi baseado em grande parte no trabalho de [Possi, 2012]. Porém com algumas diferenças.

A implementação do agente representando as células APC é um exemplo. No trabalho de [Possi, 2012], foi utilizada a linguagem de programação Java e este agente no modelo deste autor é uma classe abstrata, onde todos os agentes derivados desta classe herdam as funcionalidades.

Neste trabalho não foi implementado este agente, pois na linguagem de programação C não é possível fazer herança. Sendo assim, os métodos comuns aos agentes foram implementados no ambiente. Por este motivo, não foi necessário constar este agente neste modelo.

3.9.4.9 Macrophage Cells

De acordo com Folcik et al. [2007], este agente é considerado o principal da imunidade inata celular. A implementação deste agente se baseou no agente *Macrophage* apresentado no trabalho de [Possi, 2012].

Quando o *TissuePortal* detecta a presença da substância MK1 na *Tissue*, ele cria os agentes *Macrophage* nesta zona. Este procedimento simula a atração do macrófago para o sítio da infecção [Folcik et al., 2007].

A Figura 3.15 ilustra o comportamento do agente *Macrophage*. Pois, ao ser criado na *Tissue*, este agente se movimenta em direção à maior concentração da substância PK1, até localizar a região da infecção. Ao encontrá-la com a presença da substância *NECROSIS*, o agente assume o estado pró-inflamatório, pois a presença da substância acima indica que células estão sendo danificadas.

No estado pró-inflamatório o *Machophage* libera substância pró-inflamatória MK1. Este procedimento é responsável por recrutar outros elementos da imunidade. Ainda no estado pró-inflamatório o macrófago pode fagocitar células mortas, patógenos PAMP-positivos e complexos antígenos-anticorpo, marcados para serem removidos.

Após fagocitar alguns elementos, o macrófago passa a apresentar seus antígenos através do complexo de histocompatibilidade MHCII e têm seu estado alterado para ativado. Ao encontrar um linfócito T que reconheça os antígenos, o macrófago tem seu tempo de vida como ativo aumentado. Do mesmo modo, o linfócito T também aumenta seu tempo de vida.

O macrófago tanto na fase pró-inflamatório como ativo, tem seu tempo de vida estipulado pelo valor do parâmetro ***MACROPHAGELIFETIME***. Caso o mesmo não receba fatores para aumentar sua sobrevivência, é eliminado da simulação, através de um procedimento chamado morte celular programada.

3.9.4.10 Dendritic Cell

Este agente foi baseado no agente *Dendritic* apresentado por Possi [2012]. Tem como função principal a patrulha do tecido em busca de antígenos presentes em zona de perigo, as chamadas (*danger signals*). Quando antígenos são encontrados nesta região, os mesmos são capturados e apresentados aos linfócitos da imunidade inata. Segundo Possi [2012], agentes *Dendritic* só fagocitam elementos na zona de perigo e deste modo simula a tolerância periférica abstrata.

A Figura 3.16 ilustra o comportamento do agente *Dendritic*. Durante a criação da zona *Tissue*, a mesma adiciona uma informação na caixa de mensagem que indica a criação do agente *Dendritic*. Portanto, na primeira execução da simulação após a criação da *Tissue* são criados os agentes *Dendritic*. Este patrulha o tecido a procura da substância PK1, emitida por células estressadas. Em contato com a PK1, o agente *Dendritic* fica ativo e atua neste local (da infecção), fagocitando antígenos. No entanto, com o passar do tempo, este agente migra para o linfonodo e aguarda ser contatado pelos linfócitos.

No *Lymphnode*, o agente *Dendritic* possui uma estratégia para aumentar a cobertura de área. Esta estratégia consiste em afastar uns dos outros. Isto faz com que aumentem as chances de serem encontrados por um linfócito.

Por meio do mecanismo MHCII, este agente apresenta seus antígenos aos linfócitos. Portanto, cada vez que um linfócito reconhece os antígenos, o tempo de vida do agente *Denditric* é aumentado na zona *Lymphnode*. Caso o tempo de vida chegue ao fim, a célula é eliminada da simulação, pela apoptose.

De acordo com Folcik et al. [2007], este agente possui uma característica especial, pois usa a Vizinhança de Moore com raio igual a 2. Portando, enquanto outros agentes verificam seus 8 vizinhos, este verifica 24 posições ao seu entorno à procura de antígenos.

3.9.4.11 BCell

Este agente representa o linfócito B que é considerado a principal célula da imunidade inata. Foi baseado no agente ***BCell*** apresentado por Possi [2012].

Os agentes ***BCell*** se originam da zona *BoneMarrow*, onde adquirem suas especificidades e são realizadas as verificações referentes a tolerância central para averiguar se os mesmos não reconhecerão antígenos próprios.

A Figura 3.17 ilustra o comportamento deste agente. Pois após ser criado na *BoneMarrow*, migra para o *Lymphnode* e permanece no estado inativo. No entanto, quando reconhece um antígeno livre ou apresentado por agente APC como a célula *Dendritic*, este agente se torna ativo e segue a substância CK1.

Quando o agente ***BCell*** encontra um ***ThCell*** ativo de mesma especificidade, diferencia-se e passa para o estado germinativo. Também chamado de linfócito B germinativo ou plasma, simulando o plasmócito.

Nesta fase o agente produz clones que são enviados para a zona *Circulation*. Nesta zona, o agente segue a substância referente à infecção (CK1), liberada pelo *TissuePortal*. Por meio deste portal, o agente é levado para zona *Tissue*.

No estado germinativo, se o ***BCell*** encontrar um agente **APC** que apresente o mesmo antígeno que o ativou, seu tempo de vida é aumentado. Caso o contato se repita por algumas vezes, este agente se torna de memória.

Quanto o agente ***BCell*** está no estado de memória, é possível que seja ativado e passe para o estado de plasmócito instantaneamente. Isto ocorre quando este agente encontra um ***ThCell*** ativo ou detecta a presença de substância CK1.

Segundo Abbas et al. [2012] os agentes ***BCell*** são os únicos que possuem a capacidade de gerar anticorpos. No entanto, neste trabalho não será implementado o agente ***Antibody*** devido ao fato de que a comparação do modelo está sendo feita com o trabalho de [Possi et al., 2011] e neste não consta anticorpo.

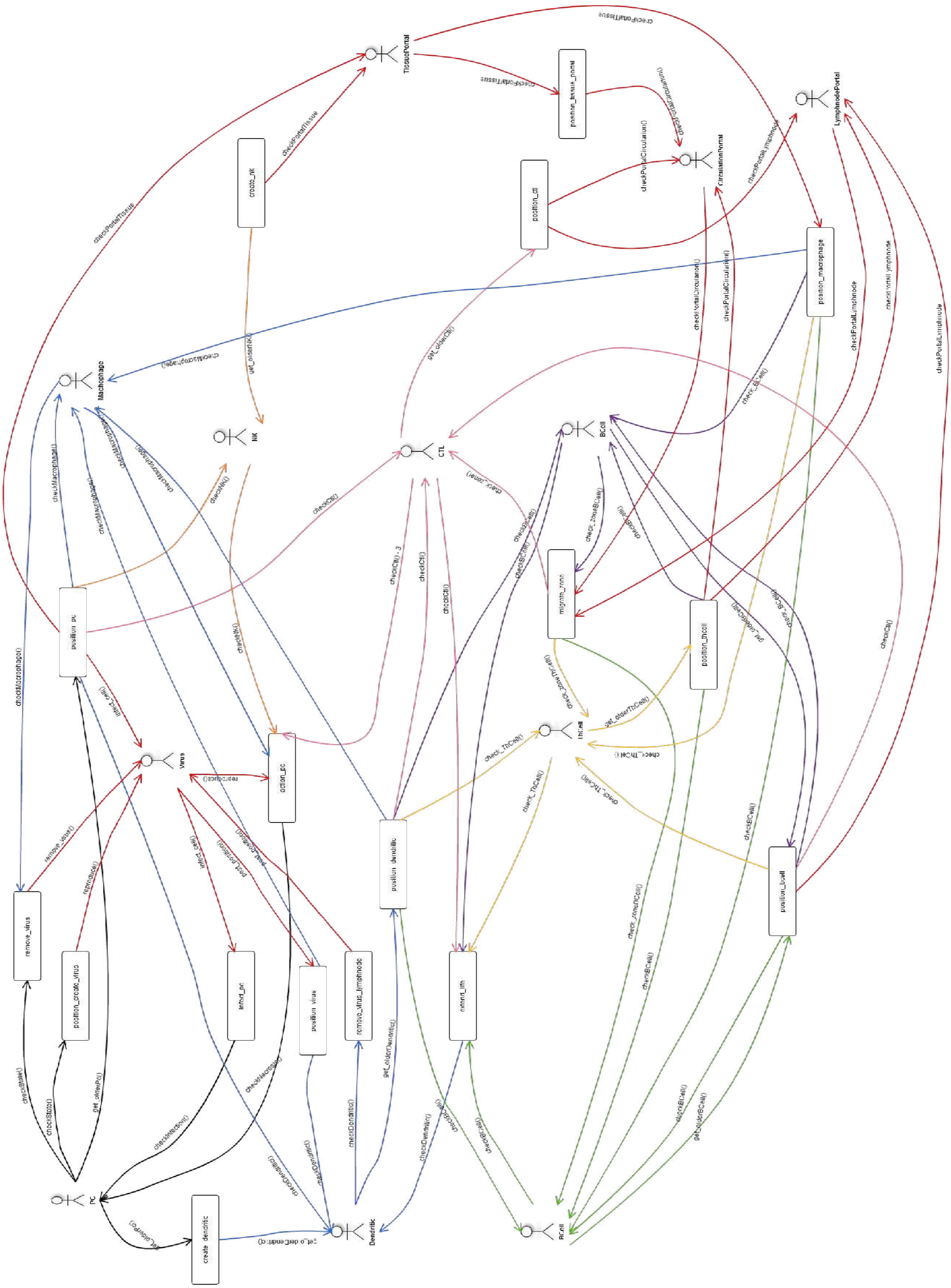

Figura 3.8. Modelo dos agentes e caixas de mensagens

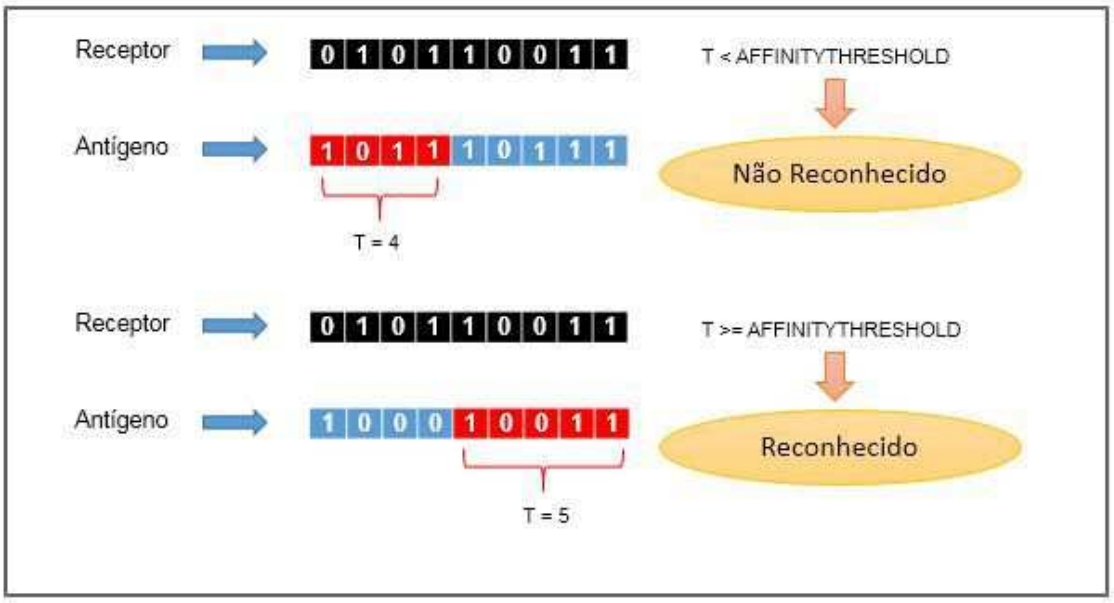

Figura 3.9. Ilustração do cálculo do Grau de Afinidade

Fonte (adaptado): [Possi, 2012]

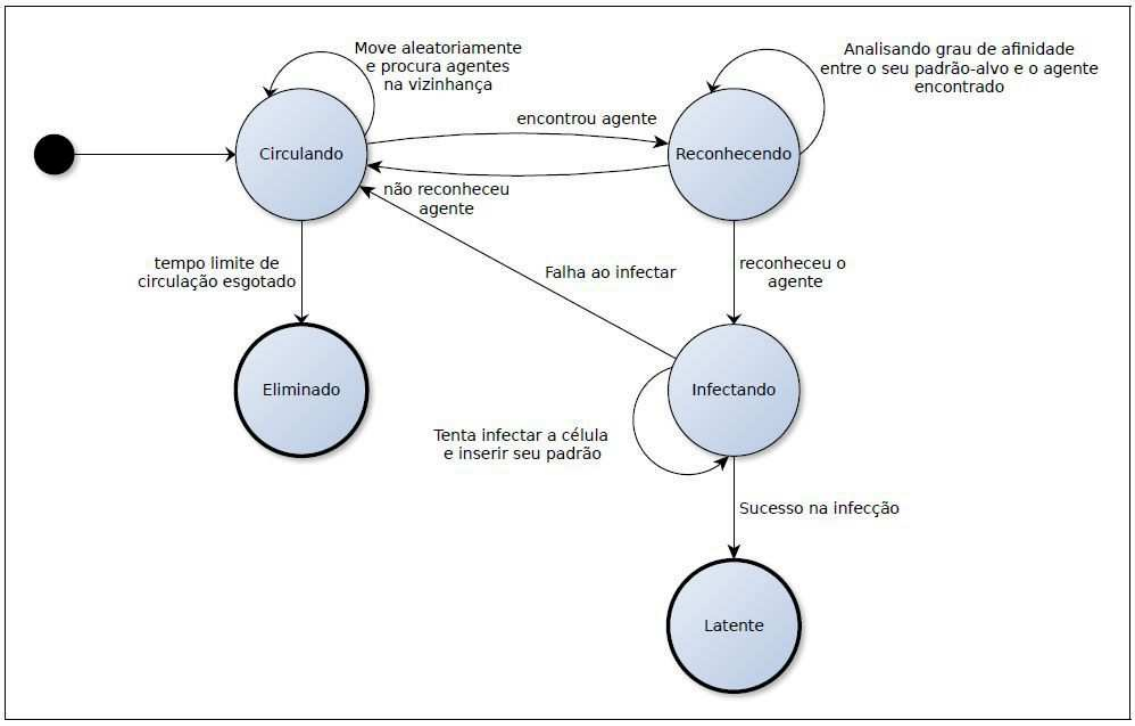

Figura 3.10. Regra do agente Vírus

Fonte: [Possi, 2012]

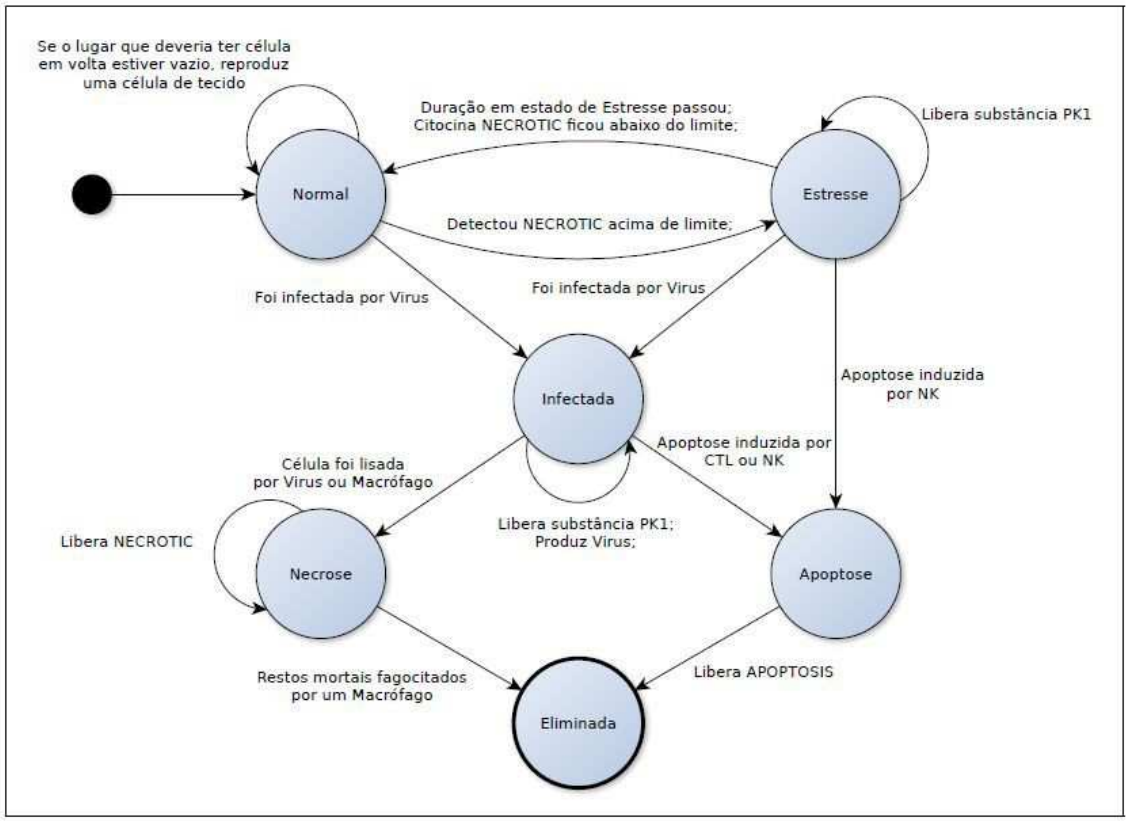

Figura 3.11. Regra do agente PC

Fonte: [Possi, 2012]

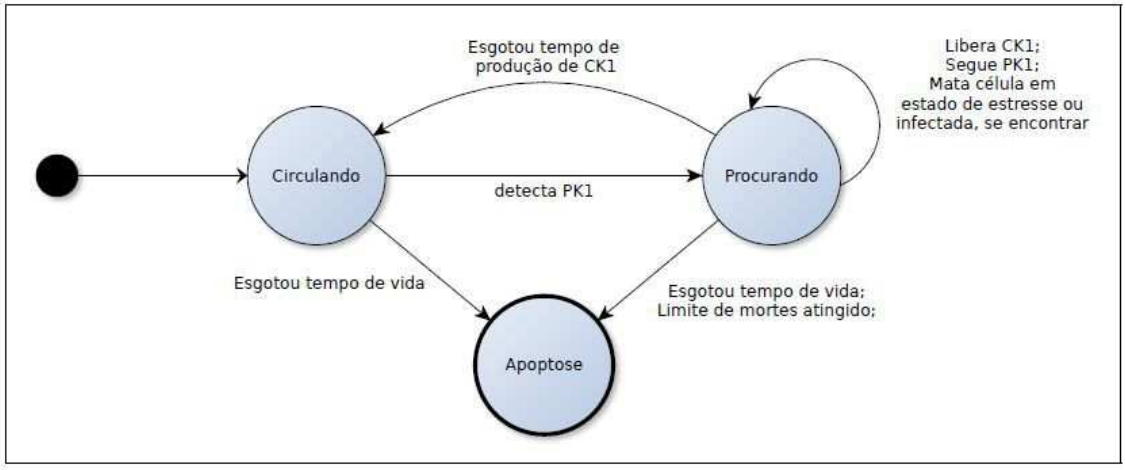

Figura 3.12. Regra do agente NK

Fonte: [Possi, 2012]

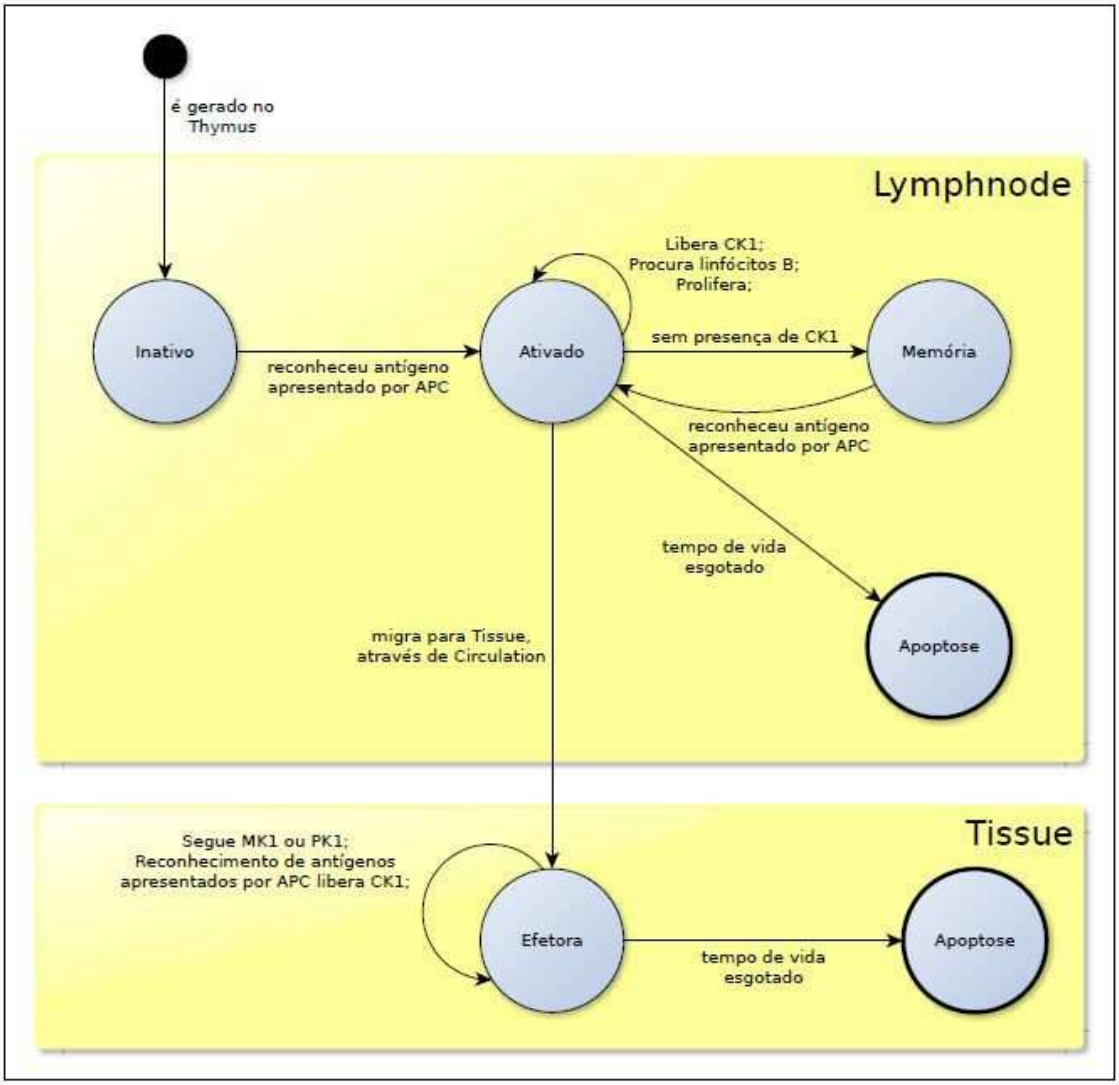

Figura 3.13. Regra do agente ThCell

Fonte: [Possi, 2012]

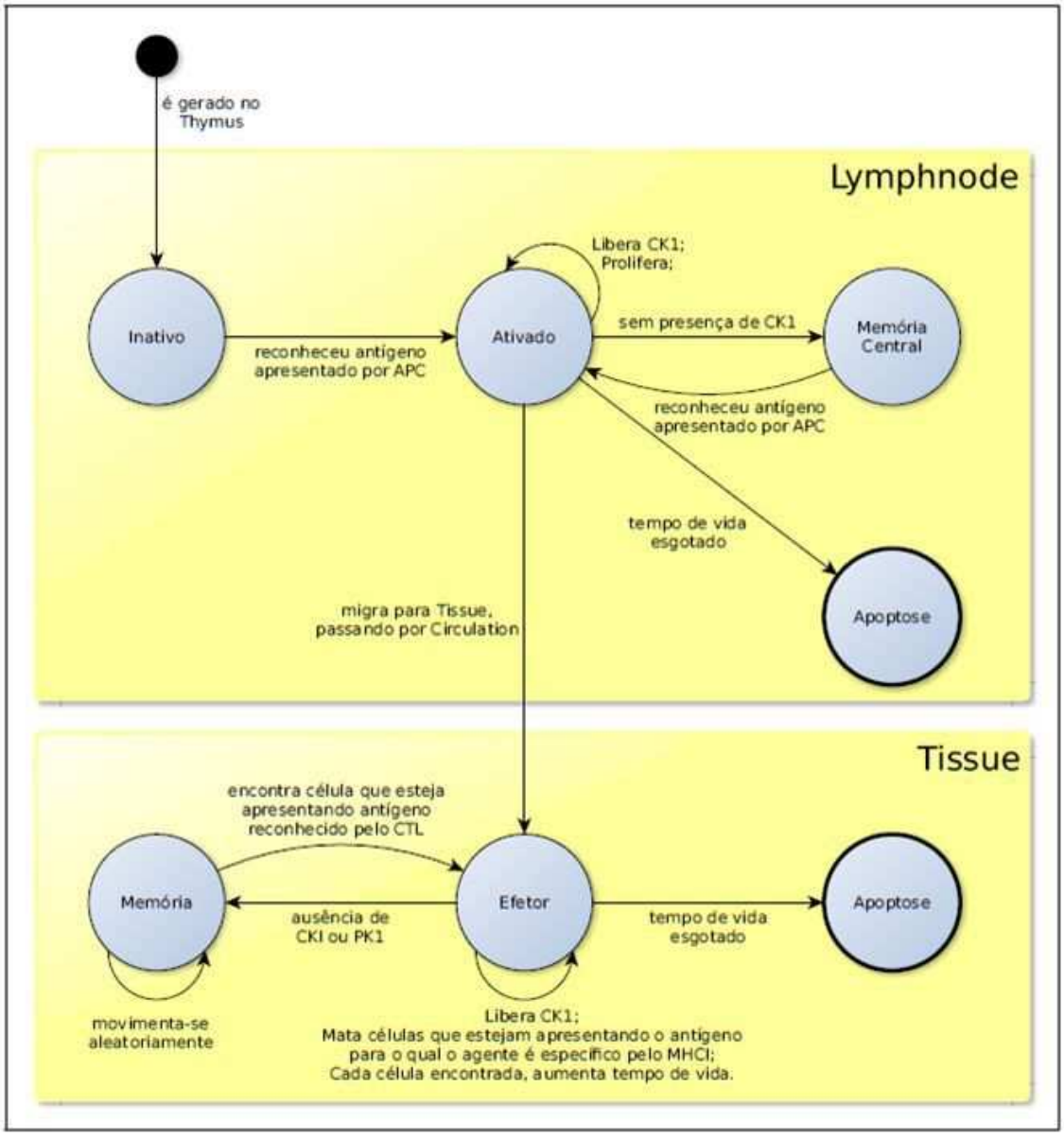

Figura 3.14. Regra do agente CTL

Fonte: [Possi, 2012]

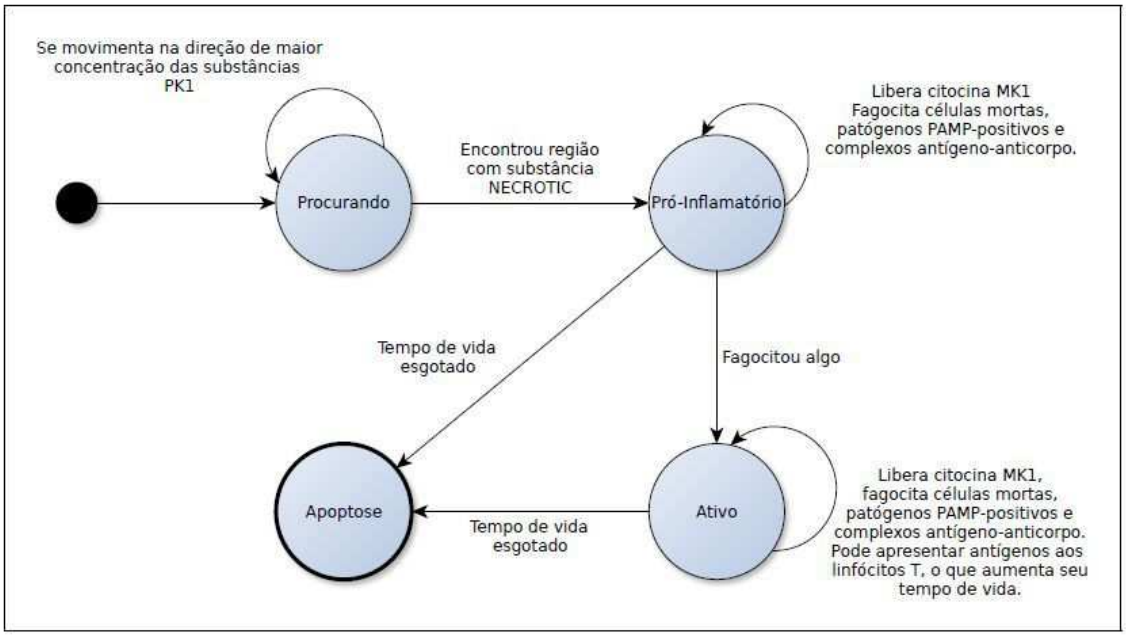

Figura 3.15. Regra do agente Macrophage

Fonte: [Possi, 2012]

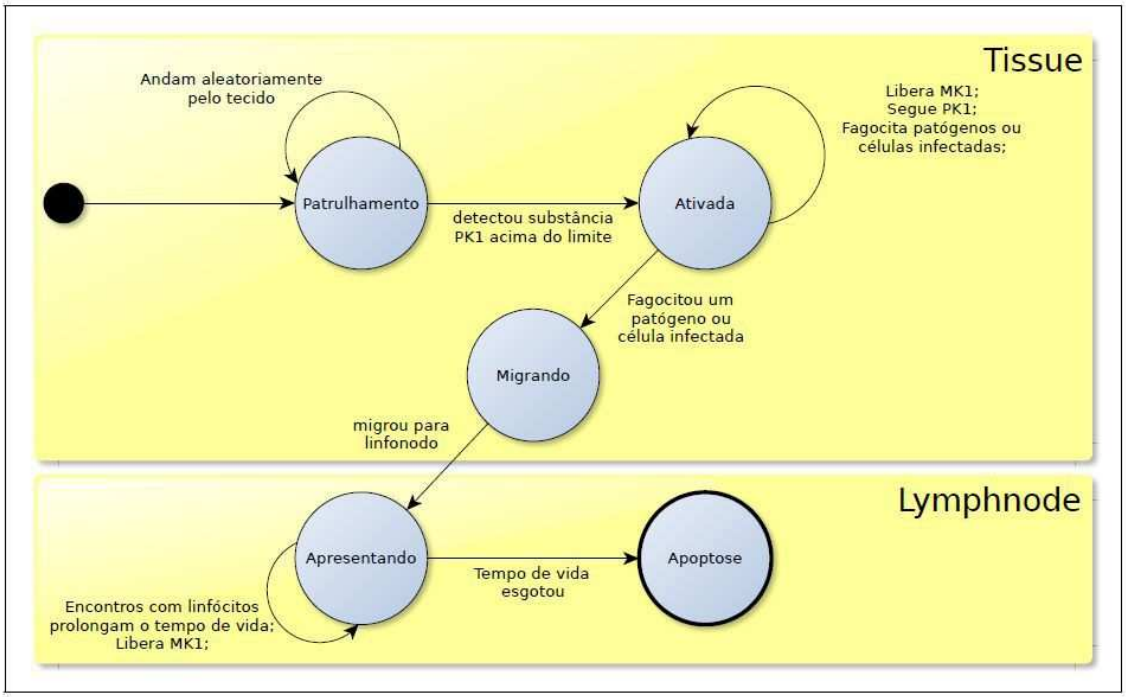

Figura 3.16. Regra do agente Dendritic

Fonte: [Possi, 2012]

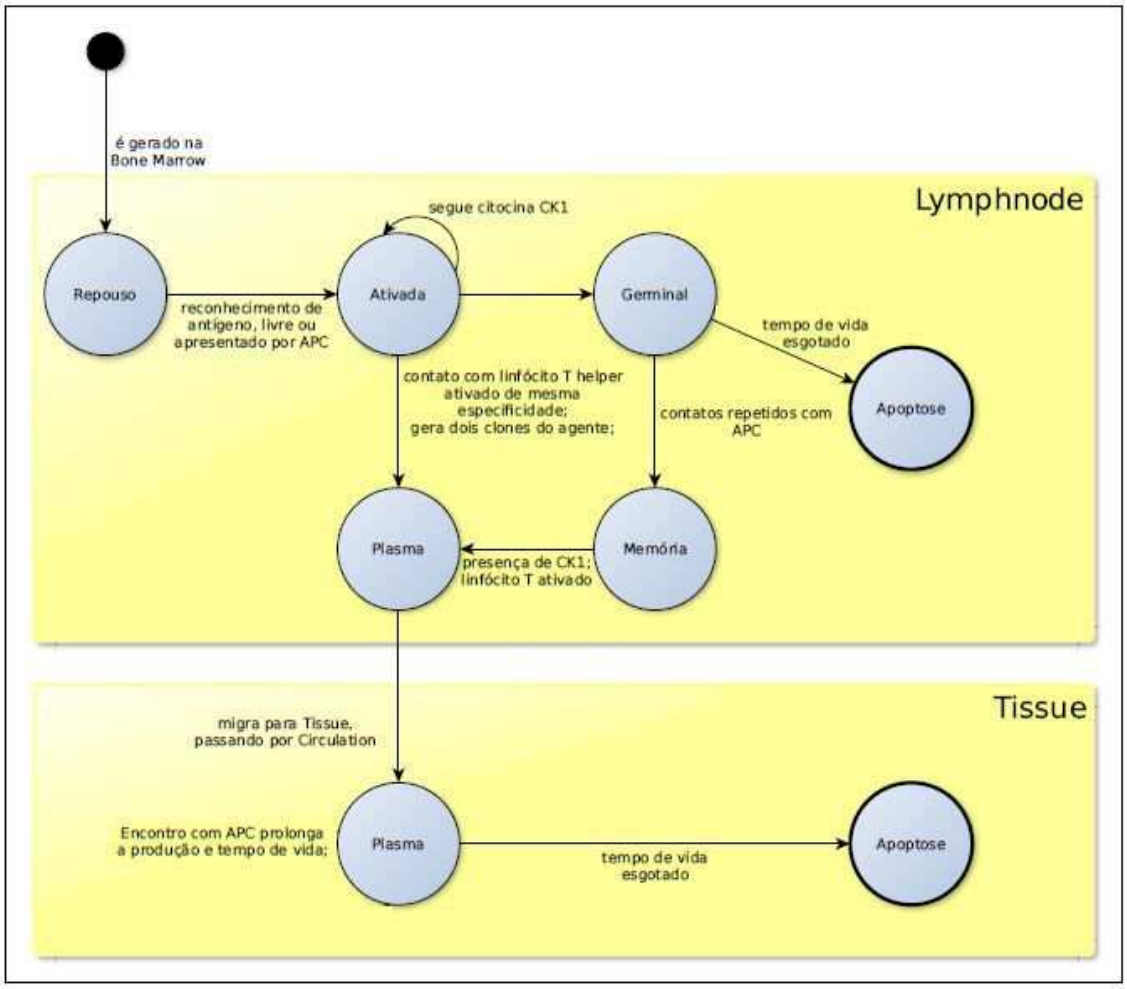

Figura 3.17. Regra do agente BCell

Fonte (adaptado): [Possi, 2012]

Capítulo 4

Resultados

Nesta seção será realizada a comparação entre a implementação de Possi et al. [2011] e o modelo descrito neste trabalho. No entanto, o trabalho de [Possi et al., 2011] foi realizado apenas na CPU. Então antes de comparar as implementações, foi realizado um estudo de caso. Com o objetivo de verificar se existe melhora no desempenho do Flame usando apenas CPU e Flame em conjunto com GPU. O estudo de caso foi executado em um problema com difusão de substância.

4.1 Simulação Flame apenas CPU e Flame em conjunto com a GPU

Foi desenvolvido um sistema multi-agente que simula: bactérias, células do sistema imunológico, bem como um conjunto de citocinas e antibióticos. O espaço é definido por um *Grid* bidimensional, cujo tamanho varia em cada conjunto de teste.

As bactérias movimentam-se aleatoriamente e reproduzem por divisão em um determinado intervalo de tempo. Caso uma célula do sistema imunológico encontre uma bactéria, esta é eliminada. Se a concentração do antibiótico no local das bactérias for maior que um valor definido por parâmetro, então estas bactérias não se reproduzirão.

Os resultados desta simulação foram comparados entre as versões Flame somente CPU e Flame+GPU, que tem este nome pelo fato do cálculo da difusão de substância ser realizado na GPU. A razão para esta comparação é tentar verificar qual arquitetura é mais adequada para utilizar em problemas desta natureza. É importante ressaltar que a implementação GPU possui um gargalo uma vez que os dados necessitam ser transferidos da memória principal para a memória da GPU.

Os testes foram realizado em uma CPU 3.60GHz 16GB RAM com microprocessador i7-4790 equipado com uma NVidia GeForce GTX 780 Ti com 2880 núcleos CUDA.

Os parâmetros de configuração para realização dos testes são:

- quantidade de camadas que estão variando entre 1, 5, 10, 15 e 20;
- número de agentes 100, 1000 e 10000;
- dimensão da matriz do tipo float 1024x1024, 2148x2048 e 16384x16384.

Em cada caso foram realizadas 1000 execuções (*ticks*). O *speedup* apresentado em cada tabela foi calculado com a equação abaixo:

$$S = ((T1\text{-}T2) \ / \ T2) * 100$$

Onde T1 representa o tempo de execução na CPU, o T2 o tempo de execução na GPU. O resultado é apresentado em percentual. Portanto, quanto maior o percentual, maior o ganho de velocidade.

4.1.1 Número de agentes

Ao observar o número de agentes por camadas é possível identificar que à medida que a quantidade de agentes aumenta, o *speedup* vai diminuindo. A seguir será mostrada uma análise de alguns casos, onde é evidenciada esta tendência.

Nos testes com 1 camada, o maior *speedup* foi o teste 16384x16384 com 100 agentes; neste constava 81,03% de *speedup*. Ao analisar o teste com 1000 agentes para este caso, o *speedup* foi reduzido para 75,62% com 10000 agentes para 1,19%. Estas variações podem ser observadas na Figura 4.1. Este mesmo comportamento está presente na Figura 4.2, onde é verificado o teste com 10 *layers*, ao analisar o caso com maior *speedup*, matriz com 1024x1024 e 100 agentes. Incialmente o *speedup* era 85,96% para 100 agentes com 1000 agentes passou para 37,55% e com 10000 o valor foi para 13,61%. Este comportamento foi constante em todas as situações analisadas; à medida que o número de agentes aumenta o *speedup* diminui. Nas Figuras 4.3 e 4.4 são apresentados os valores para 15 e 20 camadas. Apesar de faltarem alguns valores, a tendência descrita anteriormente também é mantida.

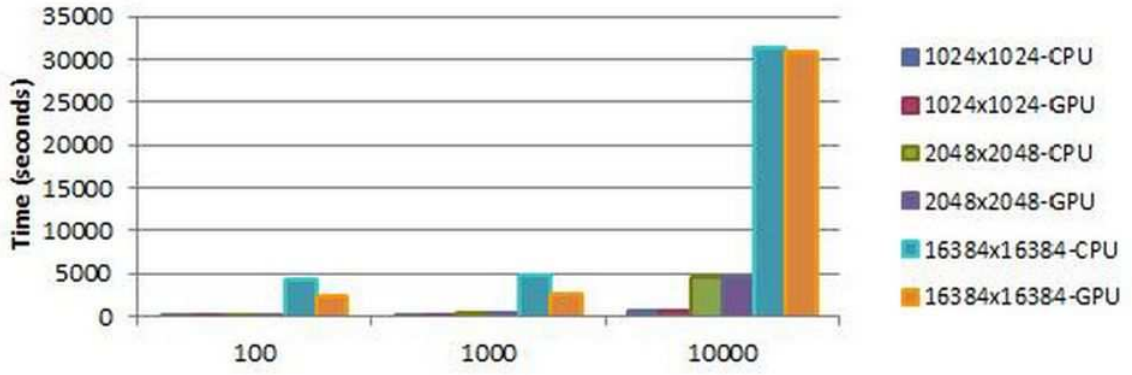

Figura 4.1. 1 Layer: Número de agentes por dimensão da matriz

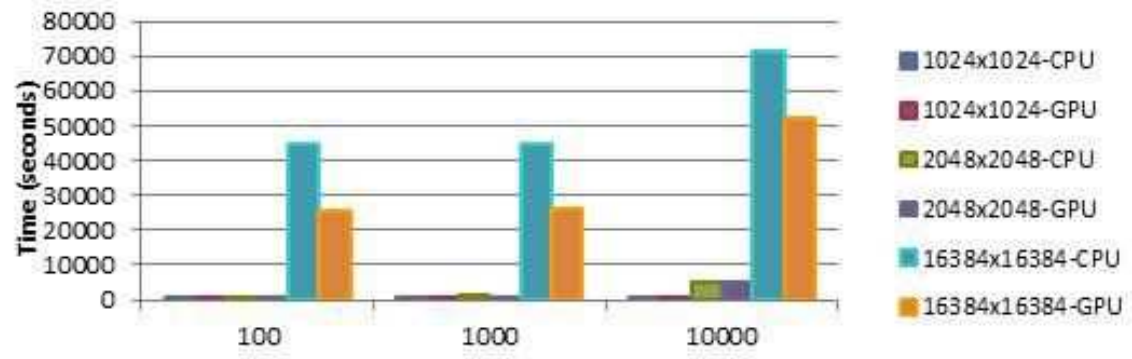

Figura 4.2. 10 Layer: Número de agentes por dimensão da matriz

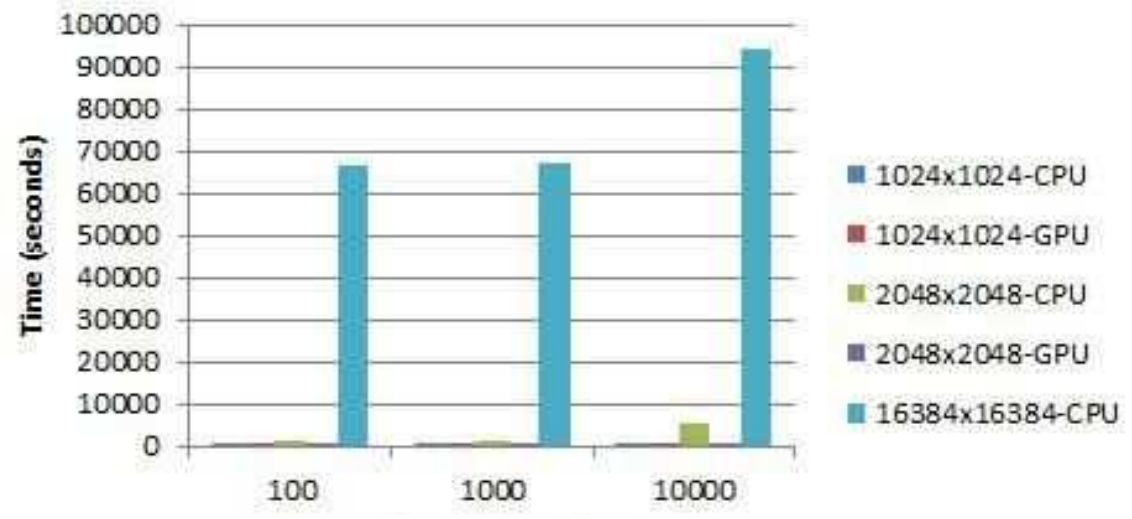

Figura 4.3. 15 Layer: Número de agentes por dimensão da matriz

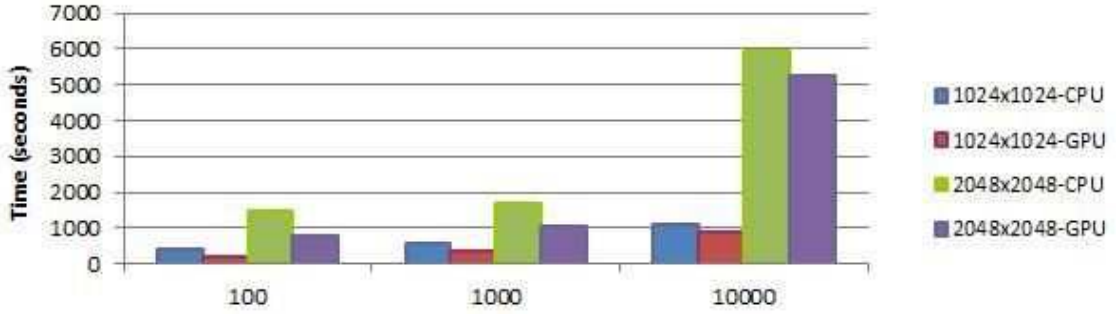

Figura 4.4. 20 Layer: Número de agentes por dimensão da matriz

4.1.2 Número de layers (camadas)

A Figura 4.5 mostra a variação no *speedup* à medida que o número de camadas aumenta. Nesta figura são apresentados os resultados das dimensões 1024x1024 com 100 agentes, 1024x1024 com 1000 agentes e 1024x1024 com 10000 agentes, com variação do número de camadas. Neste estudo de caso, pode-se destacar o exemplo 1024x1024 com 100 agentes, que obteve o resultado de *speedup* para 1 camada de 76,92%, 5 camadas 84,48%, dez camadas 85,96%, 15 camadas 92,68% e 20 camadas 97,65%. Portanto conforme mostra a Figura 4.5 para esta configuração, o aumento de camadas resultou em ganhos significativos no *speedup* em todos os casos.

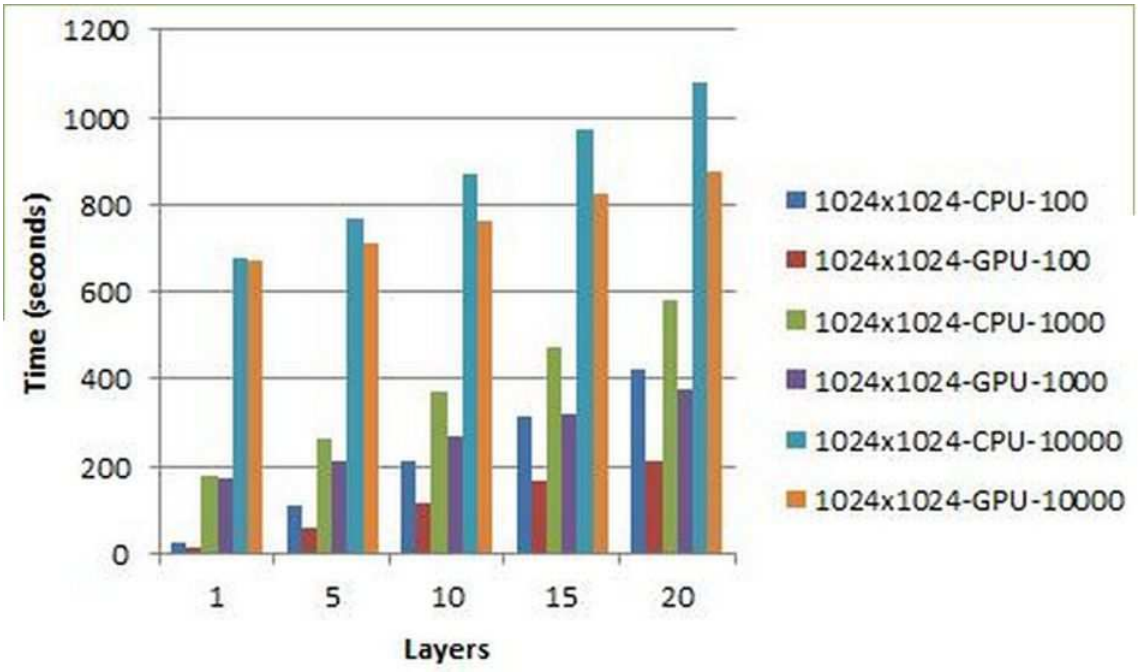

Figura 4.5. Layers por dimensão da matriz 1024x1024

A Figura 4.6 apresenta o resultado dos testes com as configurações 2048x2048-

100, 2048x2048-1000 e 2048x2048-1000. A dimensão 2048x2048-100 se destacou e os valores de speedup foram: 1 camada 78,57%, 5 camadas 79,02%, 10 camadas 82,54%, 15 camadas 81,62%, 20 camadas 84,15%. Observe que entre as camadas 10 e 15 houve um decréscimo; no entanto, na camada 20 o *speedup* melhorou. Para a dimensão 2048x2048-1000 os *speedups* foram: 1 camada 11,85%, 5 camadas 42,89%, 10 camadas 51,78%, 15 camadas 57,56% e 20 camadas 62,55%. A dimensão 2048x2048-1000 teve ganhos similares. Portanto, com esta dimensão a versão CPU-GPU apresenta ganhos significativos de desempenho.

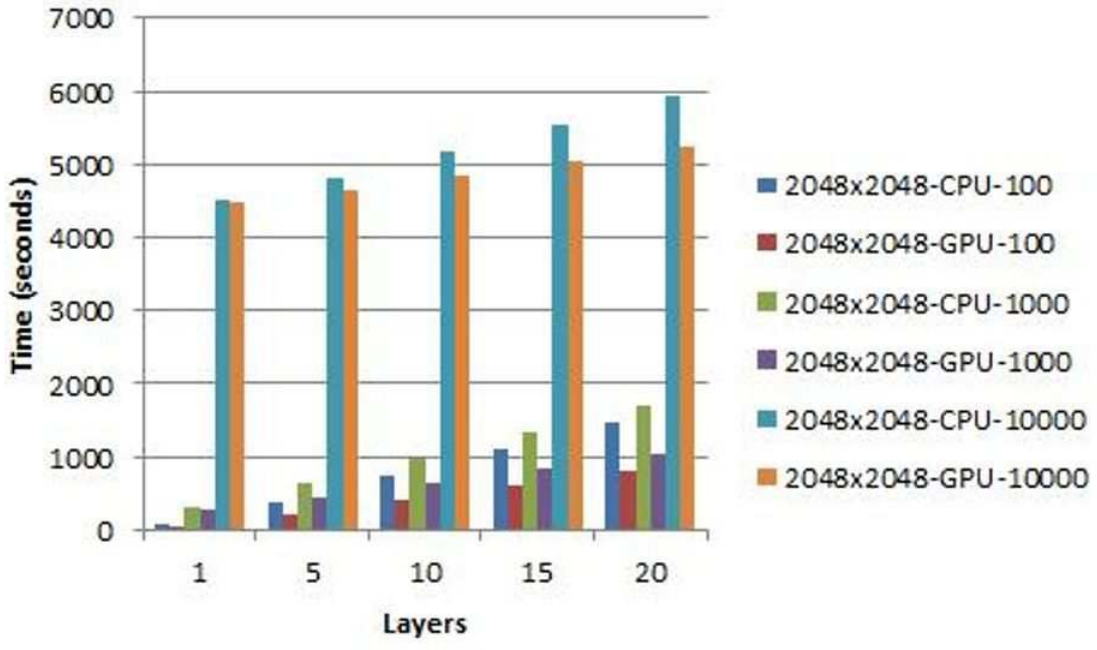

Figura 4.6. Layers por dimensions matriz 2048x2048

A Figura 4.7 apresenta os testes realizados com as dimensões 16384x16384 com 100 agentes, 16384x16384 com 1000 agentes e 16384x16384 com 10000 agentes. Para os testes com 16384x16384 com 100 agentes, o *speedup* com 1 camada foi 81,03%, com 5 camadas 77,10%, com 10 camadas 74,64%, com 15 e com 20 camadas não foi possível analisar.

Os testes 16384x16384 com 1000 agentes e 1 camada foi 75,62%, com 5 camadas 75,31%, com 10 camadas 71,84% com 15 e com 20 camadas não foram analisados.

Com a dimensão 16384x16384 com 10000 agentes e 1 camada o *speedup* foi 1,35%, com 5 camadas 22,29% com 10 camadas 37,32% com 15 e 20 camadas não foram analisados.

Neste conjunto de testes é importante observar que as dimensões 16384x16384 100 agentes e 16384x16384 1000 agentes o *speedup* foi decrescente. Porém com a

configuração 16384x16384 10000 agentes o *speedup* foi crescente com uma melhora significativa.

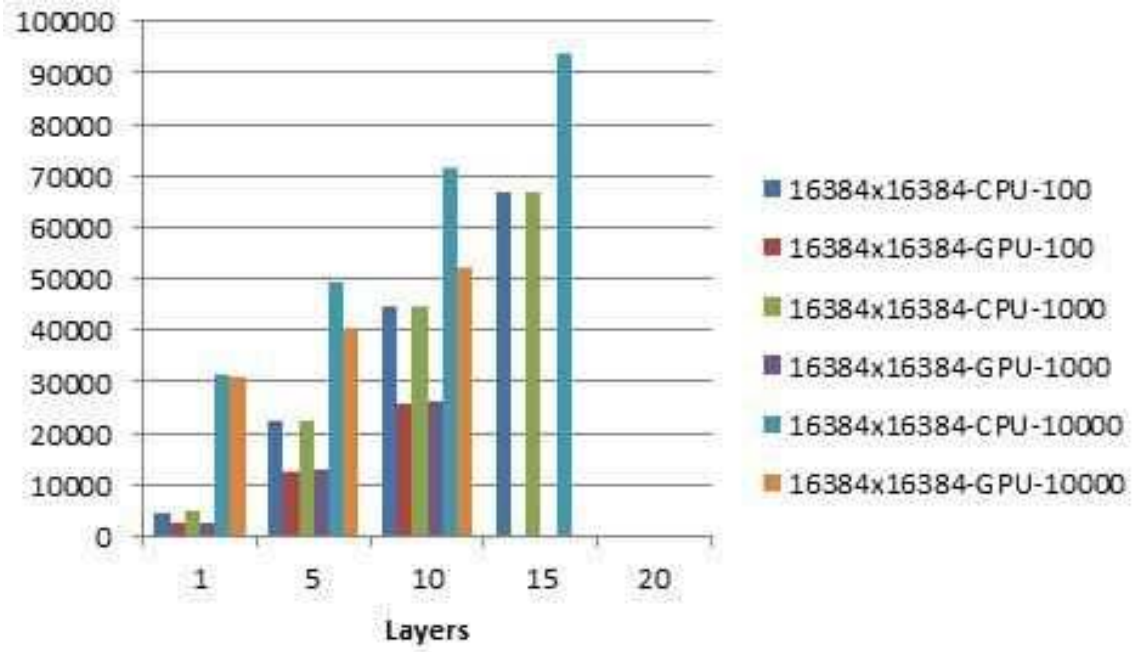

Figura 4.7. Layers por dimensions matriz 16384x16384

Os resultados apresentados mostram que utilizar o FLAME em conjunto com a GPU tem um melhor *speedup* quando comparado com a versão FLAME somente CPU. A versão FLAME+GPU foi superior em todos os casos testados. No entanto, é importante observar o teste com 20 camadas e matriz com a dimensão 1024x1024, pois este teve o melhor *speedup* de 97.65

Todos os resultados coletados durante os testes estão presentes nas Tabelas 4.1, 4.2, 4.3, 4.4, 4.5. Estes foram utilizados para gerar os gráficos e descrever as informações presentes neste tópico.

4.2 Simulação do SI

Nesta seção será apresentado o resultado da comparação entre o *AutoSimmune* e o Flame+GPU que é a implementação do modelo apresentado na seção 3.1. Os parâmetros de configuração utilizados nos dois modelos foram mantidos, exceto a dimensão das camadas de dados, já que a GPU trabalha com dimensão múltipla de 8. Então as dimensões das matrizes de difusão foram alteradas para o próximo número múltiplo de 8. Exemplo: a camada de circulação no *AutoSimmune* é 250 no Flame+GPU 256.

Tabela 4.1. Número de camadas = 1 - Número de agentes por dimensão da matriz (tempo em segundos).

Dimensão	Agente	Somente CPU	CPU+GPU	Speedup %
1024x1024	100	23	13	**76,92**
2048x2048		75	42	**78,57**
16384x16384		4446	2456	**81,03**
1024x1024	1000	180	170	**5,88**
2048x2048		321	287	**11,85**
16384x16384		4726	2691	**75,62**
1024x1024	10000	679	671	**1,19**
2048x2048		4498	4464	**0,76**
16384x16384		31382	30965	**1,35**

Tabela 4.2. Número de camadas = 5 - Número de agentes por dimensão da matriz (tempo em segundos).

Dimensão	Agente	Somente CPU	CPU+GPU	Speedup %
1024x1024	100	107	58	**84,48**
2048x2048		367	205	**79,02**
16384x16384		22228	12551	**77,10**
1024x1024	1000	265	214	**23,83**
2048x2048		643	450	**42,89**
16384x16384		22505	12837	**75,31**
1024x1024	10000	765	713	**7,29**
2048x2048		4795	4632	**3,52**
16384x16384		49200	40233	**22,29**

Tabela 4.3. Número de camadas = 10 - Número de agentes por dimensão da matriz (tempo em segundos).

Dimensão	Agente	Somente CPU	CPU+GPU	Speedup %
1024x1024	100	212	114	**85,96**
2048x2048		732	401	**82,54**
16384x16384		44537	25502	**74,64**
1024x1024	1000	370	269	**37,55**
2048x2048		979	645	**51,78**
16384x16384		44798	26070	**71,84**
1024x1024	10000	868	764	**13,61**
2048x2048		5159	4829	**6,83**
16384x16384		71478	52052	**37,32**

Para A execução dos testes foram definidas faixas de *ticks* que são: 500, 1000, 1500, 2000, 2500, 3000. Em cada caso, foram realizadas 10 execuções, os valores foram adicionados nas Tabela 6.1 e 6.2. Portanto a Tabela 4.6 apresenta um consolidado com a média de cada faixa. Nesta mesma tabela, a coluna referente ao *speedup*

Tabela 4.4. Número de camadas = 15 - Número de agentes por dimensão da matriz (tempo em segundos).

Dimensão	Agente	Somente CPU	CPU+GPU	Speedup %
1024x1024	100	316	164	**92,68**
2048x2048		1097	604	**81,62**
16384x16384		66660	0	**0,00**
1024x1024	1000	474	317	**49,53**
2048x2048		1344	853	**57,56**
16384x16384		66904	0	**0,00**
1024x1024	10000	974	822	**18,49**
2048x2048		5525	5036	**9,71**
16384x16384		93960	0	**0,00**

Tabela 4.5. Número de camadas = 20 - Número de agentes por dimensão da matriz (tempo em segundos).

Dimensão	Agente	Somente CPU	CPU+GPU	Speedup %
1024x1024	100	421	213	**97,65**
2048x2048		1464	795	**84,15**
16384x16384		0	0	**0,00**
1024x1024	1000	579	377	**53,58**
2048x2048		1710	1052	**62,55**
16384x16384		0	0	**0,00**
1024x1024	10000	1079	876	**23,17**
2048x2048		5925	5233	**13,22**
16384x16384		0	0	**0,00**

mostra quantas vezes o Flame+GPU foi mais rápido que o *AutoSimmune*. Para este cálculo, foi utilizada a equação abaixo. Optou-se por este modelo de cálculo de *speedup*, pois não é viável usar percentual quando os valores são autos:

$$S = (T1\text{-}T2) \ / \ T2$$

Os testes foram executados em um computador equipado com sistema operacional Linux Ubuntu 14.04, com CPU 3.60GHz 16GB RAM processador i7-4790 com uma NVidia GeForce GTX 780 Ti e 2880 núcleos CUDA.

A tabela 4.6 mostra os resultados dos experimentos entre o *AutoSimmune* versão que utiliza o *Repast Simphony* versão 1.2 e o Flame+GPU, pode-se observar o menor *speedup* na comparação entre as duas abordagens que é 16,13, isto quando foram executados 500 *ticks*. Este valor mostra que o Flame+GPU é 16,13 vezes mais rápido que o *AutoSimmune*. Nesta mesma tabela, é exibido o melhor *speedup* que é 25,88, ou seja o Flame-GPU executou 25,88 vezes mais rápido que o *AutoSimmune*.

Tabela 4.6. Média dos experimentos AutoSimmune e Flame+GPU

Tempos em (ms)			
Ticks	**AutoSimmune**	**Flame+GPU**	**Speedup**
500	113854,6	6645,40	16,13
1000	236120,2	11031,70	20,40
1500	352582,6	14671,00	23,03
2000	476920,7	20006,20	22,84
2500	603317,3	24000,00	24,14
3000	735206,8	27354,70	25,88

Em todos os casos, o Flame+GPU teve o desempenho superior ao *AutoSimmune*. No entanto, os resultados apontam uma tendência para os testes com os dois *frameworks*, pois a medida em que o número de *tiks* aumenta, o tempo de processamento também aumenta. Porém no Flame+GPU o crescimento desta linha de tendência é mais lenta, este comportamento pode ver verificado na Figura 4.8.

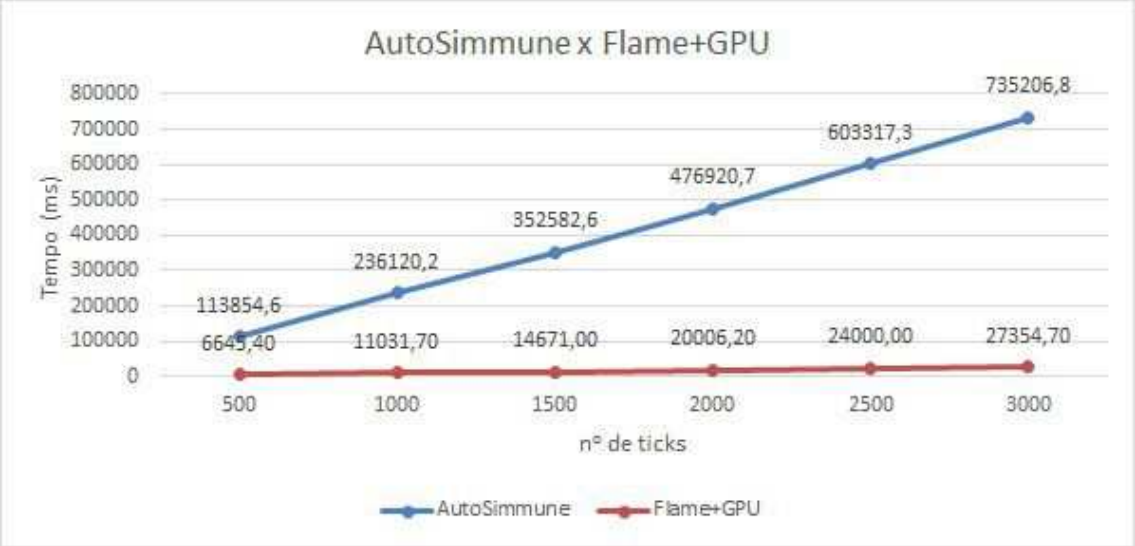

Figura 4.8. AutoSimmune x Flame+GPU

Capítulo 5

Conclusões

De acordo com a proposta inicial do trabalho, foi elaborado um modelo de simulação para o sistema imunológico SI humano. A implementação deste modelo foi realizada utilizando o *framework* Flame que é um HPC. Este *framework*, por ser genérico, não possui estrutura para auxiliar a modelagem do SI. Esta característica tornou a implementação mais flexível, uma vez que se pode criar as estruturas desejadas. No entanto, aumentou a complexidade de implementação.

Um ponto fundamental desta pesquisa foi a integração do Flame com a GPU que chamamos de Flame+GPU, uma vez que a utilização do FlameGPU foi descartada.

Ao comparar a implementação do Flame+GPU com o *AutoSimmune*, pode-se observar que o Flame+GPU apresenta resultados até 25,88 vezes melhores que o *AutoSimmune*. Este foi o maior *speedup* apresentado nos testes, resultado obtido para 3000 *ticks*. Já o menor resultado foi 16,13 vezes melhor que o *AutoSimmune*, para a configuração de 500 *ticks*. Entretanto, os resultados mostraram que à medida em que a quantidade de *tiks* aumenta, o *speedup* favorável ao Flame+GPU também aumenta. Então para simulações maiores, a diferença tende a aumentar.

Portanto, os objetivos específicos foram atendidos e mostram que nesta situação (simulação do SI), programação paralela, uso de GPU e *framework* HPC, dão melhores resultados, quando comparados com *frameworks* de ABMS sequenciais tradicionais.

Com efeito, esta pesquisa confirma a proposição de Collier & North [2012], na qual se afirma que quando as simulações exigem grande poder computacional e manipulam muitos dados, estas devem usar *frameworks* HPC.

Espera-se que a partir de agora novas funcionalidades sejam adicionadas neste modelo, e que este trabalho contribua para o avanço das pesquisas realizadas pelo

grupo de Modelagem e Simulação do Sistema Imunológico (ModeSimmune) da Universidade Federal de Viçosa.

5.0.1 Trabalhos Futuros

Abaixo são apresentadas algumas sugestões de trabalhos futuros:

- Adicionar novas funcionalidades neste modelo, pois já foram realizados outros trabalhos no grupo de estudo da Universidade Federal de Viçosa e o trabalho de Possi et al. [2011], já foi estendido;
- Utilizar mais de uma placa gráfica (GPU). Ainda nesta linha, verificar a possibilidade de usar GPU para mais atividades, pois à medida que novas funcionalidades são adicionadas, a possibilidade de usar GPU é ampliada;
- Desenvolver uma ferramenta que possibilite visualizar as camadas de dados física, pois esta informação não é facilmente obtida, uma vez que, os dados estão no XML. Isto ajudará o usuário em suas verificações;
- Desenvolver uma ferramenta que informe o andamento da simulação em tempo de execução. Existe um projeto chamado EURACE que disponibiliza esta funcionalidade ao término;
- Propor aos desenvolvedores do Flame que adicionem uma forma para definirmos que um método será executado na GPU. Hoje não existe esta opção, ou se faz manualmente como apresentado neste trabalho ou se utiliza o FlameGPU;
- Criar algumas funcionalidades específicas para este problema (simulação do SI), como camada de dados física (*tissue, circulation, lymphonode* e outras). Estas seriam nativas do simulador e o procedimento de difusão seria automático, após a execução de cada *tick*.

Referências Bibliográficas

Abbas, A. K.; Lichtman, A. H. & Pillai, S. (2012). *Imunologia celular e molecular*. Elsevier Brasil, 7 edição.

Bastos, C. A.; Oliveira, A. d. P.; Gomes, A. P.; de Araújo Possi, M.; Santana, L. A.; Cerqueira, F. R. & Siqueira-Batista, R. (2013). Simulação do sistema imunológico por meio de sistemas multiagentes: um estudo da resposta imune na glomerulonefrite pós-infecciosa (gnpe) por streptococcus pyogenes.

Bordini, R. H.; Vieira, R. & Moreira, Á. F. (2001). Fundamentos de sistemas multiagentes. In *Anais do XXI Congresso da Sociedade Brasileira de Computação (SBC2001)*, volume 2, pp. 3--41.

Chin, A. L.; Worth, A. D.; Greenough, A. C.; Coakley, S.; Holcombe, M. & Kiran, M. (2012). Flame: An approach to the parallelisation of agent-based applications. *Work*, 501:63259.

Collier, N. & North, M. (2012). Parallel agent-based simulation with repast for high performance computing. *Simulation*, p. 0037549712462620.

Delves, P. J.; Martin, S. J.; Burton, D. R. & Roitt, I. M. (2011). *Roitt's essential immunology*, volume 20. John Wiley & Sons.

Floreano, D. & Mattiussi, C. (2008). *Bio-inspired artificial intelligence: theories, methods, and technologies*. MIT press.

Folcik, V. A.; An, G. C. & Orosz, C. G. (2007). Theoretical biology and medical modelling. *Theoretical Biology and Medical Modelling*, 4:39.

Hübner, J. F.; Bordini, R. H. & Vieira, R. (2004). Introdução ao desenvolvimento de sistemas multiagentes com jason. *XII Escola de Informática da SBC*, 2:51--89.

Li, X.-h.; Wang, Z.-x.; Lu, T.-y. & Che, X.-j. (2009). Modelling immune system: principles, models, analysis and perspectives. *Journal of Bionic Engineering*, 6(1):77--85.

Macal, C. M. & North, M. J. (2009). Agent-based modeling and simulation. In *Winter simulation conference*, pp. 86--98. Winter Simulation Conference.

Murphy, K. (2010). *Imunobiologia de Janeway*. Artmed Editora, 7 edição.

NVIDIA (2015). *CUDA C Programming Guide*. NVIDEA. DisponÃvel em: http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html Acessado em: 01/10/2015.

Peakman, M. & Vergani, D. (2011). *Imunologia básica e clínica*. Elsevier Brasil, 2 edição.

Pereira, F. M. Q. (2011). Técnicas de otimização de código para placas de processamento gráfico. In *XXXI Congresso da SBC Jornada de Atualização da Informática*.

Possi, M.; Oliveira, A.; Chaves, C.; Cerqueira, F. & Arroyo, J. (2011). An in-silico immune system model for investigating human autoimmune diseases. In *XXXVII Conferencia Latinoamericana de Informática (XXXVII CLEI)*. sn.

Possi, M. d. A. (2012). *Uma ferramenta para simulação do sistema imunológico, atravÃ©s de sistemas multiagentes: um caso de estudo da autoimunidade*. PhD thesis, Universidade Federal de viçosa - UFV, www.ppgcc.ufv.br. Dissertação de mestrado.

Rapin, N.; Lund, O.; Bernaschi, M. & Castiglione, F. (2010). Computational immunology meets bioinformatics: the use of prediction tools for molecular binding in the simulation of the immune system. *PLoS One*, 5(4):e9862.

Richmond, P.; Walker, D.; Coakley, S. & Romano, D. (2010). High performance cellular level agent-based simulation with flame for the gpu. *Briefings in bioinformatics*, 11(3):334--347.

Silva, C. C. d.; de Paiva Oliveira, A.; de Araújo Possi, M.; Cerqueira, F. R.; Gomes, A. P.; Santana, L. A. & Siqueira-Batista, R. (2012). Immune system simulation: Modeling the mast cell. In *Bioinformatics and Biomedicine (BIBM), 2012 IEEE International Conference on*, pp. 1--4. IEEE.

Siqueira-Batista, R.; Gomes, A. P.; Azevedo, S. F. M.; Vitorino, R. R.; Mendonça, E. G. d.; Sousa, F. O. d.; Oliveira, A. d. P.; Cerqueira, F. R.; Paula, S. O. d. & Oliveira, M. G. d. A. (2010). Linfócitos t cd4+ cd25+ e a regulação do sistema imunológico: perspectivas para o entendimento fisiopatológico da sepse. *Revista Brasileira de Terapia Intensiva*, 24(3):294--301.

Siqueira-Batista, R.; Gomes, A. P.; Bastos, C. A.; Paula-Santos, E. d.; Azevedo, S. F. M. d.; Mendes, T. A.; Oliveira, A. d. P.; Cerqueira, F. R.; Paula, S. O. d. & Geller, M. (2015). The complement system: importance in clinical practice. *RBM rev. bras. med*, 72(3).

Sousa, F. O. d.; de Paiva, A. O.; Santana, L. A.; Cerqueira, F. R.; Siqueira-Batista, R. & Gomes, A. P. (2014). Predicting the occurrence of sepsis by in silico simulation. In *Nature-Inspired Computation and Machine Learning*, pp. 486--498. Springer.

Weiss, G. (1999). *Multiagent systems: a modern approach to distributed artificial intelligence*. The MIT press.

Wooldridge, M. (2001). *An introduction to multiagent systems*. Wiley. com.

Worth, D.; Chin, L. & Greenough, C. (2012). *FLAME tutorial examples: a simple SIR infection model*. STFC.

Zia, K.; Riener, A.; Farrahi, K. & Ferscha, A. (2012). A new opportunity to urban evacuation analysis: very large scale simulations of social agent systems in repast hpc. In *Principles of Advanced and Distributed Simulation (PADS), 2012 ACM/IEEE/SCS 26th Workshop on*, pp. 233--242. IEEE.

Capítulo 6

APÊNDICE A: TABELA GERAL

A tabela 6.1 apresenta os valores das execuções dos testes como *framework* Flame+GPU. Já a tabela 6.2 refere-se aos experimentos com o Autosimmune. Em ambas as tabelas os tempos de execução são apresentados em milissegundos.

Tabela 6.1. Execução de experimentos Flame+GPU

Flame+GPU tempo em (ms)						
Execução	**500**	**1000**	**1500**	**2000**	**2500**	**3000**
1	6573	11128	16319	22485	24000	30080
2	6590	11067	16371	21028	24000	30005
3	6633	10989	14395	17410	23000	29984
4	6517	11004	14220	19717	24000	26184
5	6895	11033	15593	19418	29000	23989
6	6625	11097	13945	21914	19000	23014
7	6524	10664	14541	21347	24000	23683
8	6509	10596	14772	20275	24000	26385
9	6978	10663	14636	16241	24000	31630
10	6610	12076	11918	20227	25000	28593
Média	**6645,40**	**11031,70**	**14671,00**	**20006,20**	**24000,00**	**27354,70**

Tabela 6.2. Execução de experimentos AutoSimmune

AutoSimmune tempo em (ms)						
Execução	**500**	**1000**	**1500**	**2000**	**2500**	**3000**
1	114208	235071	316001	458466	626904	837807
2	112995	238607	344915	465551	578596	741207
3	111312	234105	342166	495715	564870	769355
4	112579	231358	335537	473161	655351	720256
5	114215	236690	346388	481957	610763	744255
6	114536	236063	389281	481525	630207	702826
7	113422	250786	351052	472793	627280	704731
8	116808	233072	368191	470849	566126	702411
9	114143	233134	368497	483011	589778	729687
10	114328	232316	363798	486179	583298	699533
Média	**113854,60**	**236120,20**	**352582,60**	**476920,70**	**603317,30**	**735206,80**

Capítulo 7

APÊNDICE B: MODELO DE AGENTE

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE xmodel SYSTEM "http://eurace.cs.bilgi.edu.tr/XMML.dtd">
<xmodel version="2">
    <name>Simple PC model</name>
    <version>01</version>
    <description></description>
    <environment>
        <functionFiles>
            <file>functionspc.c</file>
        </functionFiles>
    </environment>
    <agents>
        <xagent>
            <name>PC</name>
            <description></description>
%            <memory>
                <variable>
                    <type>int</type>
                    <name>id</name>
                    <description></description>
                </variable>
```

```
...

</memory>
<functions>

    <function>
        <name>get_olderPC</name>
        <description></description>
        <currentState>startPC</currentState>
        <nextState>1PC</nextState>
        <outputs>
          <output>
            <messageName>create_dendritic</messageName>
          </output>

        </outputs>
        <outputs>
            <output>
                <messageName>position_pc</messageName>
            </output>
        </outputs>
    </function>

    <function>
        <name>checkInfection</name>
        <description></description>
        <currentState>1PC</currentState>
        <nextState>2PC</nextState>
        <inputs>
            <input>
                <messageName>infect_pc</messageName>
            </input>
        </inputs>
```

```
                </function>

                ...

            </functions>
        </xagent>
    </agents>
    <messages>

        ...

        <message>
            <name>action_pc</name>
            <description>Necrosis cell PC</description>
            <variables>
                <variable>
                    <type>int</type>
                    <name>id</name>
                    <description></description>
                </variable>
                <variable>
                    <type>int</type>
                    <name>action</name>
                    <description></description>
                </variable>
            </variables>
        </message>

    </messages>
</xmodel>
```

Capítulo 8

APÊNDICE C: FUNÇÕES

```
#include <math.h>
#include "../header.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include "../global.h"
#include "../PC_agent_header.h"

extern float getCitokineValue(int x, int y, int zoneCitokine);

extern void releaseCitokine(int x, int y, int zoneCitokine);

/* Need count of the number of agents */
extern int GLOBAL_num_agents;

/* Increment age and length of time agent has been sick.*/
int get_olderPC() {

        /* Do some accounting in the first iteration */
        if (iteration_loop == 1) {
                createAgentsTissue();

                // Add cell dendritic in box of create
                int i = 0;
```

```
                for (i = 0; i < DENDRITICCELLSCOUNT; i++) {
                        add_create_dendritic_message(get_id());
                }
        }

        /* Read agent memory */
        int id = get_id(), type = get_type(), x = get_x(),
        y = get_y(), z = get_y();
        char * pattern = get_pcSelfPattern();
        add_position_pc_message(id, x, y, z, type, pattern,
        get_idVirus(), get_currentstate(), get_isinfected());
        return 0; /* Otherwise all is OK */
}
...
//Method brand PC cells infected.
int checkInfection() {

        /* Loop through all messages Cell PC */
        START_INFECT_PC_MESSAGE_LOOP

        if (infect_pc_message->id == get_id()) {
                int state = get_currentstate();
                if (get_isinfected() == 0 && state
                        != TISSUESTATE_NECROSIS && state
                        != TISSUESTATE_APOPTOSIS) {
                        set_isinfected(1);
                        set_idVirus(infect_pc_message->idvirus);
                        changeState(TISSUESTATE_STRESSED);
                }
        }

        FINISH_INFECT_PC_MESSAGE_LOOP
        return 0;
}
...
```