Universidade Federal de Juiz de Fora
Instituto de Ciências Exatas
Bacharelado em Ciência Da Computação

Análise do impacto proveniente do emprego de uma rede rápida e desligamento do Ecc da GPU, no desempenho de uma simulação do sistema imune inato

Thiago Marques Soares

JUIZ DE FORA
DEZEMBRO, 2014

Análise do impacto proveniente do emprego de uma rede rápida e desligamento do Ecc da GPU, no desempenho de uma simulação do sistema imune inato

Thiago Marques Soares

Universidade Federal de Juiz de Fora

Instituto de Ciências Exatas

Departamento de Ciência da Computação

Bacharelado em Ciência Da Computação

Orientador: Marcelo Lobosco

JUIZ DE FORA

DEZEMBRO, 2014

Análise do impacto proveniente do emprego de uma rede rápida e desligamento do Ecc da GPU, no desempenho de uma simulação do sistema imune inato

Thiago Marques Soares

MONOGRAFIA SUBMETIDA AO CORPO DOCENTE DO INSTITUTO DE CIÊNCIAS EXATAS DA UNIVERSIDADE FEDERAL DE JUIZ DE FORA, COMO PARTE INTEGRANTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE BACHAREL EM CIÊNCIA DA COMPUTAÇÃO.

Aprovada por:

Marcelo Lobosco
Doutor

Bernardo Martins Rocha
Doutor

Rodrigo Weber dos Santos
Doutor

JUIZ DE FORA

05 DE DEZEMBRO, 2014

Resumo

O desenvolvimento de modelos computacionais que simulam a resposta imune do corpo humano é uma tarefa complexa. Um dos problemas enfrentados é a grande quantidade de recursos computacionais necessários para executar essas simulações. Neste trabalho foi realizada a paralelização de um código sequencial que simula a formação de um abscesso em uma seção tridimensional do tecido. O arcabouço utilizado na paralelização do código baseou-se em um modelo proposto na literatura que faz uso de múltiplas GPUs em um ambiente distribuído. O trabalho também avaliou o impacto no desempenho da aplicação do emprego de uma rede de baixa latência (*Infiniband*) e do desligamento do ECC (*Error Correction Check*) presente nas GPUs. Acelerações de até 956 vezes foram obtidas quando a versão paralela do código foi executada em um ambiente com *Infiniband* e ECC desligado.

Palavras-chave: Sistema Humano Imune Inato, Computação Paralela, Avaliação de Desempenho, Infiniband, ECC.

Abstract

The development of computational models that simulate the immune response of the human body is a complex task. One of the main issues is the huge computational resources required to execute theses simulations. In this work, we have implemented a parallel version of the sequential code that simulates formation of an abscess in a three-dimensional section of a tissue. The framework used to implement the parallel version of the code is based on a model found in the literature that uses multiple GPUs on a distributed environment. This work also evaluated the performance impact of using *Infiniband* network as well as turning off the Error Correction Code present on the GPUs. Speedups up to 956 were obtained by the parallel version when using *Infiniband* and turning off ECC.

Keywords: Human Innate Immune System, Parallel Computing, Performance Evaluation, Infiniband, ECC.

Agradecimentos

Aos meus pais e parentes, pelo apoio e pela paciência ao longo dos anos.

Ao professor Marcelo Lobosco, pela orientação, paciência e apoio oferecido desde o princípio.

Aos meus amigos de curso: Igor Russo, Bruno Marques, Marcelo Marques e Victor Patrocínio, pessoas que tornaram esses quatro anos na faculdade divertidos.

A Micael Peters e Alexandre Bittencourt pelo auxílio fornecido.

A todos os amigos que, mesmo que de maneira indireta, ajudaram na conclusão deste projeto de faculdade, especialmente Evandro Tavares, Paulo Portugal, Wesley Dornellas, Reginaldo Ferreira, Vítor Moreira e Áthos Souza.

"Talvez não tenha conseguido fazer o melhor, mas lutei para que o melhor fosse feito. Não sou o que deveria ser, mas Graças a Deus, não sou o que era antes".

Martin Luther King

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviações

CUDA	*Compute Unified Device Architecture*
CPU	*Central Processing Unit*
ECC	*Error Correction Check*
EDO	Equações Diferenciais Ordinárias
EDP	Equações Diferenciais Parciais
GPU	*Graphical Processing Units*
GPGPU	*General-Purpose Graphics Processing Unit*
MPI	*Message Passing Interface*
OpenMP	*Open Multi Processing*
RAM	*Random Access Memory*
SIH	Sistema Imune Humano
UVA	*Unified Virtual Addressing*

1 Introdução

1.1 Motivação

O sistema imunológico (ou sistema imune) é de fundamental importância na manutenção da vida de diversas espécies de organismos. Sua principal função é atuar na identificação e eliminação de quaisquer agentes patogênicos externos que tentam invadir um organismo vivo. Caso o agente patogênico seja bem sucedido em sua tentativa de invasão, ele pode ocasionar uma série de doenças no organismo ou até mesmo levá-lo à morte. A resposta imune inata a uma infecção causada por um patógeno (bactéria, protozoário, vírus ou fungos) é uma resposta inflamatória. Durante uma resposta inflamatória pode ocorrer a formação de um padrão conhecido como abscesso. Um abscesso é uma coleção de pus (neutrófilos) que acumula em um tecido como consequência de um processo inflamatório em resposta a uma infecção. É uma região caracterizada por elevadas concentrações de células mortas, tecido morto, fluidos corporais, bactérias e neutrófilos (Pigozzo et al., 2013).

A modelagem matemática pode ajudar a entender grandes processos complexos, em particular processos com fortes interações acopladas e dependentes no tempo, como ocorre na formação do abscesso. Nos últimos anos, modelos matemáticos do SIH (Sistema Imune Humano) têm obtido relativo sucesso em elucidar os mecanismos por trás da resposta imune a determinadas doenças, sendo importantes, por exemplo, na definição de estratégias terapêuticas. Pigozzo et al. (2013) propôs um modelo matemático que visa reproduzir o processo inicial de formação de um abscesso. O modelo simula o comportamento das células e moléculas ao decorrer do tempo em uma seção unidimensional de tecido, considerando, para tal, aspectos como a difusão das células e substâncias no tecido bem como o fenômeno da quimiotaxia. A quimiotaxia é um fenômeno extremamente importante para uma resposta imune eficiente. As substâncias quimioatraentes como, por exemplo, as citocinas pró-inflamatórias, são responsáveis por guiar o movimento das células do SIH em direção ao local de maior concentração de antígenos.

A simulação do modelo proposto por Pigozzo et al. (2013), se estendida para um domínio espacial tridimensional (3D), requer um elevado poder de processamento, especialmente quando considera-se que a resposta imune dura vários dias. Este fato torna difícil a realização de experimentos *in silico*, dado a demora na execução do código. Neste contexto a computação paralela pode ser utilizada para reduzir o tempo total de computação (Pacheco et al., 2011).

1.2 Objetivos

Este trabalho tem por objetivo principal estender o modelo unidimensional originalmente proposto por Pigozzo et al. (2013) para um domínio tridimensional, bem como paralelizar o modelo resultante, de modo a reduzir o tempo de computação. Para a paralelização do código foi utilizado como arcabouço o modelo computacional proposto por Xavier (2013), que faz uso de múltiplas GPUs (*Graphical Processing Units*) em um agregado (*cluster*) computacional.

No entanto, o uso de GPUs em um ambiente distribuído como o agregado computacional introduz uma nova questão: a comunicação de dados entre as unidades de processamento. A escolha de uma melhor solução de interconexão para um agregado é muito importante, já que um ponto crítico para o desempenho é justamente o custo de comunicação entre os nós. O emprego de uma estrutura de conexão que possua alta vazão de dados e baixa latência, como a *Infiniband*, pode contribuir para a redução dos custos de comunicação.

Um outro fator que pode ser explorado para aumentar ainda mais o desempenho da aplicação é o ECC (*Error Correction Check*). O *hardware* da GPU utiliza parte da memória dedicada para realizar operações de detecção e correção de erros, cujo custo tanto em termos de espaço de memória quanto de tempo de execução das operações afeta o desempenho da execução da aplicação. Ao desabilitar esta funcionalidade é então possível reduzir ainda mais o custo computacional da aplicação.

Assim, este trabalho também tem por objetivo avaliar o impacto no tempo de computação da versão paralela do simulador da rede *Infiniband* e da desabilitação do ECC na GPU.

1.3 Trabalhos Relacionados

Pigozzo (2011) começou a explorar as características da computação paralela implementando 3 versões do seu código sequencial em C para simulação do funcionamento do sistema imune inato: a) a primeira usou o modelo de memória compartilhada (versão OpenMP - *Open Multi Processing*); b) a segunda o modelo de passagem de mensagem (versão MPI - *Message Passing Interface*); c) e a terceira uma versão híbrida (OpenMP + MPI). Depois seu código foi estendido para 3D e implementado em CUDA (*Compute Unified Device Architecture*), arquitetura desenvolvida pela NVIDIA, fazendo uso da Unidade de Processamento Gráfico (*Graphics Processing Unit*) (Rocha, 2012). Visando aumentar ainda mais o desempenho e a eficiência do código, ele foi estendido para fazer uso de múltiplas GPUs localizadas em uma mesma máquina (Xavier et al., 2014).

O trabalho de Rocha (2012) apresenta um modelo baseado em EDPs (Equações Diferenciais Parciais) distinto do modelo proposto originalmente por Pigozzo (2011): a) acrescentou-se no modelo espaço em forma de um cubo (3D) ao invés de linha (1D), b) as condições iniciais e de borda foram adaptadas para o modelo 3D e c) a difusão da quimiotaxia é aplicada em três direções (x, y e z). Rocha et al. (2012) realizou ainda a análise de sensibilidade do modelo, utilizando CUDA para acelerar a obtenção dos resultados. Os resultados de seus experimentos referentes a análise de sensibilidade revelam que a versão CUDA foi responsável por acelerar a execução da aplicação em 276 vezes em relação ao modelo sequencial. Em particular, o ganho computacional deve-se à quantidade enorme de dados e ao fato de não haver nenhuma dependência e/ou comunicação entre as tarefas paralelas. Para a aceleração do modelo tridimensional propriamente dito, uma primeira versão do código, ainda sem otimizações, obteve uma aceleração de cerca de 19 vezes no tempo de execução. Já com o uso de otimizações, o ganho no desempenho chegou a aproximadamente 65 vezes em relação ao código sequencial (Rocha, 2012).

Baseando-se também no trabalho de Pigozzo (2011), a dissertação de Xavier (2013) também buscou realizar uma simulação que descreve o Sistema Imunológico Humano, utilizando uma representação tridimensional do tecido humano e um conjunto de EDPs para simular a dinâmica de alguns componentes do SIH quando este é exposto ao lipopolissacarídeo (LPS). Tomando como base o modelo computacional desenvolvido por

Rocha (2012), Xavier (2013) faz uso de múltiplas GPUs, em um ambiente de agregados computacionais, o que contribuiu significativamente na redução do tempo necessário para a execução de tal modelo. Devido ao alto custo de comunicação imposto pelo uso de múltiplos dispositivos, o emprego de técnicas alternativas às transferências de dados entre eles foi realizado. No total, três abordagens distintas foram criadas para lidar com este obstáculo: a primeira delas realiza as transferências de dados entre distintas GPUs usando a memória principal como espaço intermediário de armazenamento; a segunda versão utiliza o recurso UVA (*Unified Virtual Addressing*) para que as GPUs transfiram dados diretamente entre seus espaços de endereçamento, por fim, a versão mais elaborada tenta sobrepor as transferências de dados com computação, empregando *streams* CUDA para este propósito. Através destas técnicas foi possível atingir *speedups* expressivos em relação ao modelo que emprega uma única GPU.

Em termos de desempenho, Xavier (2013) conclui que apesar dos ótimos ganhos em aceleração obtidos por cada versão implementada neste trabalho, a comunicação ainda é considerada uma das principais responsáveis pela aplicação não ter alcançado acelerações ainda melhores. É neste contexto desafiador que este trabalho apresenta uma de suas contribuições, estudando o impacto dos seguintes fatores no aumento da aceleração da aplicação:

- Tipo de rede de comunicação utilizado para interligar as máquinas (*InfiniBand* ou *Gigabit Ethernet*), e
- O impacto de um aspecto do *hardware* da GPU (ECC) no desempenho da aplicação.

1.4 Método

Para os testes realizados neste trabalho, considerou-se o modelo apresentado em Pigozzo et al. (2012) para a formação de um abscesso. O modelo descrito por Pigozzo et al. (2012) simula o comportamento temporal e espacial de dez células/moléculas do SIH em uma seção unidimensional do tecido. O presente trabalho estende o modelo para uma região tridimensional do tecido. Na implementação paralela do novo modelo resultante foram aplicados os métodos e técnicas descritas no trabalho de Xavier (2013), onde foi realizada

a paralelização da aplicação em uma abordagem híbrida utilizando CUDA, OpenMP e MPI para que esta pudesse ser executada em um ambiente de agregados computacionais com computadores que possuem múltiplas GPGPUs.

O método adotado para avaliar o impacto no desempenho consistiu em medir do tempo de execução da aplicação quando executada em duas redes distintas, *Ethernet* e *Infiniband*, e em dois cenários distintos de ECC, habilitado e desabilitado. Todas as combinações possíveis de cenários foram avaliados.

1.5 Organização do Trabalho

Este trabalho está organizado da seguinte forma. O capítulo 2 aborda o modelo matemático utilizado neste trabalho, descrevendo todas as suas equações. No capítulo 3 são apresentadas todas as tecnologias utilizadas na implementação paralela do modelo 3D. O capítulo 4 descreve a paralelização do modelo matemático. Os resultados obtidos neste trabalho estão descritos no capítulo 5. Concluindo este trabalho, o capítulo 6 traz as considerações finais e trabalhos futuros.

2 Modelo Matemático

2.1 Equações do Modelo

O modelo matemático deste trabalho é baseado nos modelos descritos anteriormente por Pigozzo et al. (2012, 2013). Em seus estudos, ele reproduz os aspectos espaciais e temporais de uma infecção bacteriana e o processo de formação de um abscesso.

O modelo matemático simula o comportamento temporal e espacial da bactéria (B), bactéria morta (BD), macrófagos não ativados (MR), macrófagos ativados (MA), neutrófilos (N), neutrófilo apoptótico (ND), citocina pró-inflamatória (CH), citocina anti-inflamatória (CA), tecido vivo (HT) e tecido morto (TD).

A relação entre os componentes acontece de acordo com o seguinte processo: neutrófilos, macrófagos não ativos e macrófagos ativos fagocitam a bactéria. O neutrófilo sofre apoptose (morte celular), o que pode ou não ser induzido pelo processo de fagocitose. Neste estado, neutrófilos apoptóticos não podem realizar a fagocitose ou produzir citocina pró-inflamatória. Consequentemente neutrófilos apoptóticos são eliminados do corpo pelos macrófagos ativos. Os neutrófilos apoptóticos eventualmente vão morrer após um período de tempo, liberando grânulos citotóxicos e enzimas de degradação no meio, causando danos no tecido através da destruição de células saudáveis. A área de infecção é limpa pelo processo de fagocitose nas células do tecido morto, realizado pelos macrófagos ativos e não ativos. Tecido saudável, em contato com bactéria, neutrófilos e macrófago ativo, produz citocina pró-inflamatória. A citocina pró-inflamatória aumenta a permeabilidade dos vasos sanguíneos, assim mais neutrófilos são recrutados para combater o tecido infectado, além de funcionar também como um atrator químico para os macrófagos não ativados, macrófagos ativos e neutrófilos. A produção de citocina pró-inflamatória é bloqueada quando o macrófago ativo ou neutrófilo entram em contato com a citocina anti-inflamatória. A ativação de macrófagos também é bloqueada pela ação da citocina anti-inflamatória, que é produzida por macrófagos ativos e macrófagos não ativados que estão em contato com neutrófilos apoptóticos (Pigozzo et al., 2012, 2013).

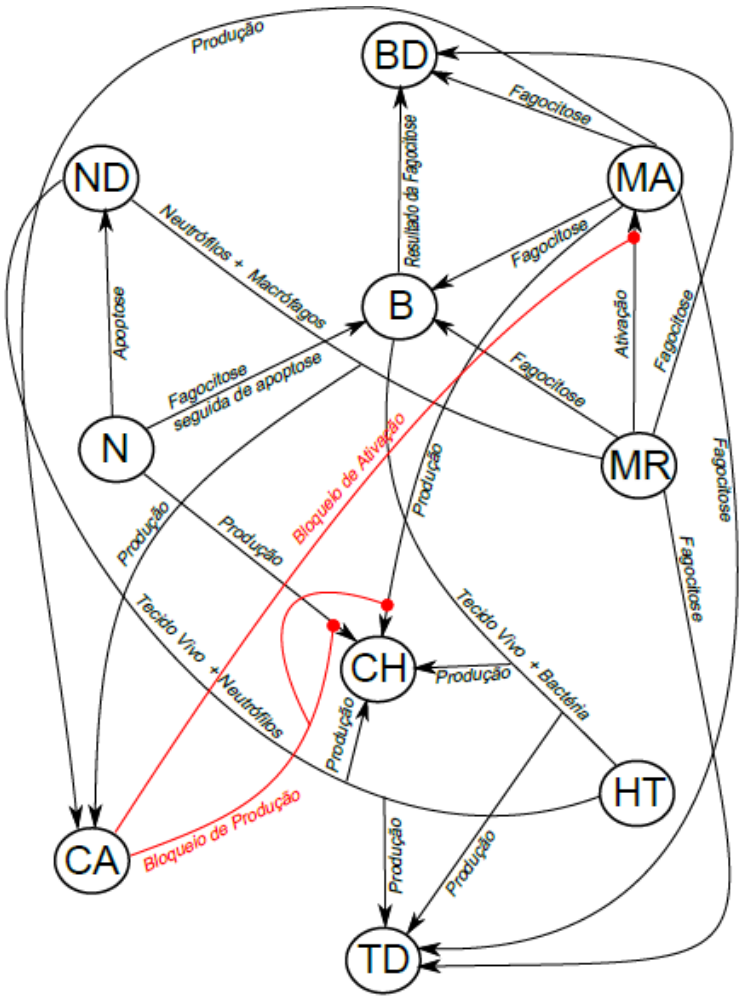

Figura 2.1: Diagrama de relações entre os componentes do modelo

Abaixo são apresentadas as equações derivadas do modelo. A equação 2.1 refere-se a equação diferencial da bactéria (B).

$$\begin{cases} maActivation = maActivationRate.MR.B/(1+\theta_{CA}.CA) \\ \\ \frac{\partial B}{\partial t} = (r_B.B - maActivation - \lambda_{N|B}.N.B - \lambda_{MR|B}.MR.B - \lambda_{MA|B}.MA.B).g(w) \\ +D_B.dif(B,w) \\ \\ B(x,y,z,0) = B_0, \frac{\partial B(.,t)}{\partial n}|_{\partial\Omega} = 0 \end{cases} \tag{2.1}$$

Nesta equação, $r_B.B$ representa o termo de reprodução da bactéria, onde r_B é a taxa de reprodução e $g(w)$ é a função da densidade total de células w em uma área discreta do espaço tridimensional em um determinado momento no tempo. O termo *ma-Activation* modela a ativação dos macrófagos não ativados. Essa ativação ocorre quando os macrófagos não ativados reconhecem algum antígeno e após esse reconhecimento é realizada a fagocitose desse antígeno.

A variável w é definida como:

$$w(x,y,z,t) = B(x,y,z,t) + BD(x,y,z,t) + N(x,y,z,t) + ND(x,y,z,t) + MR(x,y,z,t) + MA(x,y,z,t) + CH(x,y,z,t) + HT(x,y,z,t) + TD(x,y,z,t) + CA(x,y,z,t); \tag{2.2}$$

$\lambda_{N|B}.N.B$ representa a fagocitose da bactéria pelo neutrófilo, onde $\lambda_{N|B}$ é a taxa de fagocitose. $\lambda_{MR|B}.MR.B$ representa a fagocitose da bactéria pelo macrófago não ativado, onde $\lambda_{MR|B}$ é a taxa de fagocitose. $\lambda_{MA|B}.MA.B$ representa a fagocitose da bactéria pelo macrófago ativado, onde $\lambda_{MA|B}$ é a taxa de fagocitose. $D_B.dif(B,w)$ representa a difusão da bactéria, onde D_B representa o coeficiente de difusão e $dif(B,w)$ é calculado da seguinte maneira:

$$dif(B,w) = \nabla \cdot (g(w)\nabla(f(w)B) - f(w)B\nabla g(w)); \tag{2.3}$$

A função f modela a probabilidade de uma célula ser empurrada de um local devido a pressão exercida pelas células vizinhas (Painter et al., 2003, 2009). A pressão populacional é modelada pela equação de Hill (Goutelle et al., 2008). Ela aumenta com a densidade total das células w ocupando a mesma posição no espaço e tem saturação em densidade alta de células.

A função f é definida como:

$$f(w) = 1 + \alpha\frac{w}{\beta + w}, \tag{2.4}$$

onde α e β são constantes.

A função g retorna a porcentagem de espaço livre em uma área discreta do espaço tridimensional e seu uso é motivado por conceitos biológicos importantes como *quorum sensing* e *volume sensing* (Painter et al., 2002, 2003, 2009; Byrne et al., 2004; Wang et al., 2010). As células possuem um conjunto de mecanismos que detectam a densidade celular ao seu redor, e em regiões com alta densidade de células, ocorre uma modificação no comportamento dessas células, que estão sempre tentando se adaptar às condições do ambiente (Pigozzo et al., 2012).

A função g é definida como:

$$g(w) = 1 - \frac{w}{total}, \tag{2.5}$$

onde a variável *total* representa a densidade máxima que é permitida na área discreta do tecido.

A equação diferencial correspondente à bactéria morta (BD) é representada pela equação (2.6):

$$\begin{cases} \frac{\partial BD}{\partial t} = (\lambda_{N|B}.N.B + \lambda_{MR|B}.MR.B + \lambda_{MA|B}.MA.B \\ -\lambda_{MA|BD}.MA.BD - \lambda_{MR|BD}.MR.BD).g(w) \\ \\ BD(x,y,z,0) = B_0, \frac{\partial BD(.,t)}{\partial n}|_{\partial\Omega} = 0 \end{cases} \tag{2.6}$$

Os termos, $\lambda_{N|B}.N.B$, $\lambda_{MR|B}.MR.B$ e $\lambda_{MA|B}.MA.B$ foram definidos anteriormente. $\lambda_{MA|BD}.MA.BD$ representa a fagocitose da bactéria morta pelo macrófago ativo, onde $\lambda_{MA|BD}$ é a taxa de fagocitose. $\lambda_{MR|BD}.MR.BD$ representa a fagocitose da bactéria morta pelo macrófago não ativado, onde $\lambda_{MR|BD}$ é a taxa da fagocitose.

A equação diferencial correspondente ao macrófago não ativado (MR) é representada pela equação (2.7):

$$\begin{cases} P_{MR} = ((P_{MR}^{max} - P_{MR}^{min}).\frac{CH}{(CH+keq_{CH})})/(1+\theta_{CA}.CA) + P_{MR}^{min} \\ \\ source_{MR} = P_{MR}.(M^{max} - (MR+MA))) \\ \\ \frac{\partial MR}{\partial t} = (-\mu_{MR}.MR - maActivation + source_{MR}).g(w) + D_{MR}.dif(MR,w) - \\ -\chi_{MR}.chemotaxis(MR,CH,w) \\ \\ MR(x,y,z,0) = MR_0, \frac{\partial MR(.,t)}{\partial n}|_{\partial\Omega} = 0 \end{cases} \tag{2.7}$$

P_{MR} descreve o aumento na permeabilidade do endotélio aos macrófagos não ativados induzido pelas citocinas pró-inflamatórias. O cálculo deste termo envolve os seguintes parâmetros: a) *Pmax* é a taxa máxima de aumento da permeabilidade do endotélio aos macrófagos não ativados induzido pelas citocinas pró-inflamatórias, b) *Pmin* é a taxa mínima de aumento da permeabilidade do endotélio aos macrófagos não ativados induzido pelas citocinas pró-inflamatórias; e c) *keqch* é a concentração de citocina pró-inflamatória que exerce 50% do efeito máximo no aumento da permeabilidade (Pigozzo et al., 2012, 2013). O termo $source_{MR}.g(w)$ modela o termo fonte dos macrófagos, que está relacionado com o número de monócitos que vão entrar no tecido a partir dos vasos sanguíneos. Este número depende da permeabilidade do endotélio P_{MR} e do número de monócitos surgindo no sangue (M^{max}). $\mu_{MR}MR$ representa a apoptose dos macrófagos não ativados, onde μ_{MR} é a taxa de apoptose. O termo *maActivation* foi definido anteriormente. $D_{MR}.dif(MR,w)$ descreve a difusão dos macrófagos não ativados, onde D_{MR} representa o coeficiente de difusão e a função dif foi definida anteriormente. $\chi_{MR}.chemotaxis(MR,CH,w)$ representa o a quimiotaxia dos macrófagos não ativados, onde χ_{MR} é a taxa de quimiotaxia e $chemotaxis(MR,CH,w)$ é calculada da seguinte forma:

$$chemotaxis(MR,CH,w) = \nabla \cdot (MRg(w)f(w)\nabla CH); \tag{2.8}$$

A equação diferencial correspondente ao macrófago ativado (MA) é representada pela equação (2.9):

$$\begin{cases} \frac{\partial MA}{\partial t} = (-\mu_{MA}.MA + maActivation).g(w) + D_{MA}.dif(MA, w) \\ -\chi_{MA}.chemotaxis(MA, CH, w) \\ \\ MA(x, y, z, 0) = MA_0, \frac{\partial MA(.,t)}{\partial n}|_{\partial\Omega} = 0 \end{cases} \tag{2.9}$$

Note que *maActivation* foi definida anteriormente.

Os termos, $\mu_{MA}.MA$, $D_{MA}.dif(MA, w)$, e $\chi_{MA}.chemotaxis(MA, CH, w)$ modelam a apoptose do macrófago ativo (μ_{MA} é a taxa de apoptose), a difusão (D_{MA} é o coeficiente de difusão), e a quimiotaxia (χ_{MA} é a taxa de quimiotaxia), respectivamente.

A equação diferencial do neutrófilo (N) é modelada pela equação (2.10):

$$\begin{cases} P_N = ((P_N^{max} - P_N^{min}).\frac{CH}{CH + keq_{CH}})/(1 + \theta_{CA}.CA) + P_N^{min} \\ \\ source_N = P_N.(N^{max} - N) \\ \\ \frac{\partial N}{\partial t} = (-\mu_N.N - \lambda_{B|N}.B.N + source_N).g(w) + D_N.dif(N, w) - \chi_N.chemotaxis(N, CH, w) \\ \\ N(x, y, z, 0) = N_0, \frac{\partial N(.,t)}{\partial n}|_{\partial\Omega} = 0 \end{cases} \tag{2.10}$$

O termo P_N modela o aumento da permeabilidade do endotélio aos neutrófilos induzido pela citocina pró-inflamatória. P_N^{max} e P_N^{min} são, respectivamente, as taxas máxima e mínima de aumento da permeabilidade do endotélio aos neutrófilos induzida pelas citocinas pró-inflamatórias e keq_{CH} é a concentração de citocina pró-inflamatória que exerce 50% do efeito máximo no aumento da permeabilidade.

Na equação, $\mu_N.N$ representa a apoptose do neutrófilo, onde μ_N é a taxa de

apoptose. $\lambda_{B|N}.B.N$ modela a apoptose dos neutrófilos que foi induzida pelo processo de fagocitose, onde $\lambda_{B|N}$ representa a taxa desta apoptose induzida. $source_N$ representa o número de neutrófilos que vão entrar no tecido a partir dos vasos sanguíneos. Este número é dependente da permeabilidade do endotélio (P_N) e do número de neutrófilos no sangue (N^{max}).

$D_N.dif(N, w)$ modela a difusão dos neutrófilos, onde D_N é o coeficiente de difusão e a função dif foi definida anteriormente. $\chi_N.chemotaxis(N, CH, w)$ modela a quimiotaxia dos neutrófilos, onde χ_N é a taxa de quimiotaxia e a função $chemotaxis(N, CH, w)$ foi definida anteriormente.

A equação diferencial correspondente ao neutrófilo apoptótico (ND) é representada na equação (2.11):

$$\begin{cases} \frac{\partial ND}{\partial t} = (\mu_N.N + \lambda_{B|N}.B.N - \lambda_{ND|MA}.ND.MA - \mu_{ND}ND).g(w) \\ \\ ND(x, y, z, 0) = ND_0, \frac{\partial ND(.,t)}{\partial n}|_{\partial\Omega} = 0 \end{cases} \tag{2.11}$$

Na equação, os termos $\mu_N.N$ e $\lambda_{B|N}.B.N$ foram definidos anteriormente. O termo $\lambda_{ND|MA}.ND.MA$ modela a fagocitose do neutrófilo apoptótico pelo macrófago ativo, e $\lambda_{ND|MA}$ é a taxa da fagocitose. $\mu_{ND}ND$ representa a necrose do neutrófilo, onde μ_{ND} é a taxa da necrose.

A equação diferencial correspondente a citocina pró-inflamatória (CH) é apresentada na equação (2.12):

$$\begin{cases} \frac{\partial CH}{\partial t} = (-\mu_{CH}.CH + ((\beta_{CH|N}.N.B + \beta_{CH|MA}.MA.B)/(1 + \theta_{CA}.CA) \\ +\lambda_{B|HT}.B.HT + \lambda_{ND|HT}.ND.HT).(1 - CH/chInf)).g(w) + D_{CH}.dif(CH, w) \\ \\ CH(x, y, z, 0) = CH_0, \frac{\partial CH(.,t)}{\partial n}|_{\partial\Omega} = 0 \end{cases} \tag{2.12}$$

Nesta equação, $\mu_{CH}CH$ modela o decaimento da citocina pró-inflamatória, onde μ_{CH} é a taxa de decaimento. $\beta_{CH|N}.N.B$ modela a produção de citocina pró-inflamatória pelos neutrófilos, onde $\beta_{CH|N}$ é a taxa de produção. $\beta_{CH|MA}.MA.B$ modela a produção de citocina pró-inflamatória pelos macrófagos ativos, onde $\beta_{CH|MA}$ é a taxa de produção. $\lambda_{A|HT}.B.HT$ modela a produção de citocina pró-inflamatória através do contato do tecido vivo com a bactéria, onde $\lambda_{B|HT}$ é a taxa de produção. $\lambda_{ND|HT}.ND.HT$ modela a produção de citocina pró-inflamatória através do contato do tecido vivo com neutrófilos necrosados, onde $\lambda_{ND|HT}$ é a taxa de produção. Essa produção possui uma saturação que é calculada por: $1 - CH/chInf$, onde $chInf$ é a concentração máxima de citocinas pró-inflamatórias suportada pelo tecido. $D_{CH}.dif(CH, w)$ representa a difusão da citocina pró-inflamatória, onde D_{CH} é o coeficiente de difusão e a função dif foi definida anteriormente.

A equação diferencial correspondente a citocina anti-inflamatória (CA) é representada na equação (2.13):

$$\begin{cases} \frac{\partial CA}{\partial t} = (-\mu_C A.CA + (\beta_{CA|MR}.MR.ND + \beta_{CA|MA}.MA).(1 - CA/caInf)).g(w) \\ +D_{CA}.dif(CA, w) \\ \\ CA(x, y, z, 0) = CA_0, \frac{\partial CA(.,t)}{\partial n}|_{\partial\Omega} = 0 \end{cases} \tag{2.13}$$

O termo $\mu_C A.CA$ modela o decaimento da citocina anti-inflamatória, onde $\mu_C A$ é a taxa de decaimento. O termo $\beta_{CA|MR}.MR.ND$ modela a produção da citocina anti-inflamatória pelos macrófagos não ativados na presença de neutrófilos apoptóticos, onde $\beta_{CA|MR}$ é a taxa de produção. O termo $\beta_{CA|MA}.MA$ modela a produção de citocina anti-inflamatória pelos macrófagos ativados, onde $\beta_{CA|MA}$ é a taxa de produção. Essa produção possui uma saturação que é calculada pelo termo $1 - CA/caInf$, onde $caInf$ é a concentração máxima de citocina anti-inflamatória. O termo $D_{CA}.dif(CA, w)$ representa a difusão da citocina anti-inflamatória, onde D_{CA} é a taxa de difusão. A função dif foi definida anteriormente.

A equação diferencial correspondente ao tecido vivo (HT) é representada na equação (2.14):

$$\begin{cases} \frac{\partial HT}{\partial t} = (-\lambda_{HT|ND}.HT.ND - \lambda_{B|HT}.B.HT).g(w) \\ \\ HT(x,y,z,0) = HT_0, \frac{\partial HT(.,t)}{\partial n}|_{\partial\Omega} = 0 \end{cases} \tag{2.14}$$

$\lambda_{HT|ND}.HT.ND$ modela o dano causado no tecido vivo pelos neutrófilos apoptóticos, onde $\lambda_{HT|ND}$ é a taxa do dano. $\lambda_{B|HT}.B.HT$ modela o dano causado no tecido pela bactéria, onde $\lambda_{B|HT}$ é a taxa do dano.

A equação diferencial correspondente ao tecido morto (TD) é representada na equação (2.15):

$$\begin{cases} \frac{\partial TD}{\partial t} = (\lambda_{HT|ND}.HT.ND + \lambda_{B|HT}.B.HT - \lambda_{MR|TD}.MR.TD - \lambda_{MA|TD}.MA.TD).g(w) \\ \\ TD(x,y,z,0) = TD_0, \frac{\partial TD(.,t)}{\partial n}|_{\partial\Omega} = 0 \end{cases} \tag{2.15}$$

Os termos $\lambda_{HT|ND}.HT.ND$ e $\lambda_{B|HT}.B.HT$ foram definidos anteriormente. $\lambda_{MR|TD}.MR.TD$ representada a fagocitose das células do tecido morto pelos macrófagos não ativados, onde $\lambda_{MR|TD}$ é a taxa de fagocitose. $\lambda_{MA|TD}.MA.TD$ representada a fagocitose das células do tecido morto pelos macrófagos ativos, onde $\lambda_{MA|TD}$ é a taxa de fagocitose.

A seguir é apresentado um resumo de todas as equações desconsiderando a condição inicial e a condição de contorno.

Equação diferencial da bactéria (B):

$$\begin{cases} maActivation = maActivationRate.MR.B/(1+\theta_{CA}.CA) \\ \frac{\partial B}{\partial t} = (r_B.B - maActivation - \lambda_{N|B}.N.B - \lambda_{MR|B}.MR.B - \lambda_{MA|B}.MA.B).g(w) + D_B.dif(B,w) \end{cases} \tag{2.16}$$

Equação diferencial da bactéria morta (BD):

$$\left\{ \frac{\partial BD}{\partial t} = (\lambda_{N|B}.N.B + \lambda_{MR|B}.MR.B + \lambda_{MA|B}.MA.B - \lambda_{MA|BD}.MA.BD - \lambda_{MR|BD}.MR.BD).g(w) \right. \tag{2.17}$$

Equação diferencial do macrófago não ativado (MR):

$$\begin{cases} P_{MR} = ((P_{MR}^{max} - P_{MR}^{min}).\frac{CH}{(CH+keq_{CH})})/(1+\theta_{CA}.CA) + P_{MR}^{min} \\ source_{MR} = P_{MR}.(M^{max} - (MR+MA))) \\ \frac{\partial MR}{\partial t} = (-\mu_{MR}.MR - maActivation + source_{MR}).g(w) + D_{MR}.dif(MR,w) - \chi_{MR}.chemotaxis(MR,CH,w) \end{cases} \tag{2.18}$$

Equação diferencial do macrófago ativado (MA):

$$\left\{ \frac{\partial MA}{\partial t} = (-\mu_{MA}.MA + maActivation).g(w) + D_{MA}.dif(MA,w) - \chi_{MA}.chemotaxis(MA,CH,w) \right. \tag{2.19}$$

Equação diferencial do neutrófilo (N):

$$\begin{cases} P_N = ((P_N^{max} - P_N^{min}).\frac{CH}{CH+keq_{CH}})/(1+\theta_{CA}.CA) + P_N^{min} \\ source_N = P_N.(N^{max} - N) \\ \frac{\partial N}{\partial t} = (-\mu_N.N - \lambda_{B|N}.B.N + source_N).g(w) + D_N.dif(N,w) - \chi_N.chemotaxis(N,CH,w) \end{cases} \tag{2.20}$$

Equação diferencial do neutrófilo apoptótico (ND):

$$\left\{ \frac{\partial ND}{\partial t} = (\mu_N.N + \lambda_{B|N}.B.N - \lambda_{ND|MA}.ND.MA - \mu_{ND}ND).g(w) \right. \tag{2.21}$$

Equação diferencial da citocina pró-inflamatória (CH):

$$\begin{cases} \frac{\partial CH}{\partial t} = (-\mu_{CH}.CH + ((\beta_{CH|N}.N.B + \beta_{CH|MA}.MA.B)/(1+\theta_{CA}.CA) + \lambda_{B|HT}.B.HT + \lambda_{ND|HT}.ND.HT).(1 - CH/chInf)).g(w) \\ +D_{CH}.dif(CH,w) \end{cases} \tag{2.22}$$

Equação diferencial da citocina anti-inflamatória (CA):

$$\left\{ \frac{\partial CA}{\partial t} = (-\mu_C A.CA + (\beta_{CA|MR}.MR.ND + \beta_{CA|MA}.MA).(1 - CA/caInf)).g(w) + D_{CA}.dif(CA,w) \right. \tag{2.23}$$

Equação diferencial do tecido vivo (HT):

$$\left\{ \frac{\partial HT}{\partial t} = (-\lambda_{HT|ND}.HT.ND - \lambda_{B|HT}.B.HT).g(w) \right. \tag{2.24}$$

Equação diferencial do tecido morto (TD):

$$\left\{ \frac{\partial TD}{\partial t} = (\lambda_{HT|ND}.HT.ND + \lambda_{B|HT}.B.HT - \lambda_{MR|TD}.MR.TD - \lambda_{MA|TD}.MA.TD).g(w) \right. \tag{2.25}$$

2.2 Métodos Numéricos

A implementação dos métodos numéricos de Pigozzo et al. (2013) é baseada no método das diferenças finitas para a discretização espacial e no método de Euler explícito para a discretização no tempo. O método das diferenças finitas é muito usado na discretização numérica de EDPs (LeVeque et al., 2007).

A discretização do termo de quimiotaxia ($\nabla.(\chi_N N \nabla CH)$) usa um esquema *upwind* de Primeira Ordem (Roe et al., 2002). Portanto, a precisão da implementação numérica é de primeira ordem no tempo (Euler explícito) e de primeira ordem no espaço (esquema *upwind*). Os esquemas *upwind* discretizam EDPs hiperbólicas através do uso de diferenças com *bias* na direção determinada pelo sinal das velocidades características. Os esquemas *upwind* usam um *stencil* adaptativo ou sensível a solução para numericamente simular mais precisamente a direção da propagação da informação.

Outra característica importante da implementação foi a escolha de um método *one-step*, ou seja, a derivada no tempo foi discretizada utilizando a derivada para frente (*forward*) tal que a população no instante de tempo $t + 1$ só depende da população no instante de tempo t. Esta escolha permitiu diminuir consideravelmente a demanda por memória devido ao fato de que só precisamos utilizar uma posição de memória para cada posição discretizada do espaço simulado, sendo que esta posição guarda o valor da população no instante de tempo atual e é reutilizada para armazenar o próximo instante de tempo (Pigozzo et al., 2013).

3 Fundamentação Teórica

3.1 OpenMP

OpenMP (*Open Multi Processing*) é uma especificação utilizada no desenvolvimento de programas paralelos em plataformas compostas de múltiplos processadores/núcleos que compartilham informações através de uma memória compartilhada. OpenMP define um modelo portável e escalável que permite o desenvolvimento de aplicações paralelas de uma forma simples e flexível nas linguagens C, C++ e Fortran (Chandra et al., 2001).

Com o uso de OpenMP é possível criar programas paralelos em ambiente de memória compartilhada de uma maneira fácil com o uso de *threads*. Programas em OpenMP basicamente começam com um processo sequencial na CPU, chamada *thread* mestre. A *thread* mestre executa o código sequencialmente até encontrar uma região paralela. A partir daí, é criado um time de *threads* que processam simultaneamente o código delimitado pela região paralela. Após sua execução, todas as *threads* se sincronizam e terminam, restando ativa somente a *thread* mestre que originou tal conjunto (modelo *fork-join*). Uma situação em que pode-se empregar *threads* OpenMP é na gestão de recursos de uma máquina, como por exemplo múltiplas GPUs. Nesta abordagem, cada *thread* que executa na CPU é responsável pelo gerenciamento de uma única GPU, realizando funções como alocação de memória, chamada do *kernel* e transferência de dados entre esta GPU e a CPU (Xavier, 2013).

Na aplicação desenvolvida neste trabalho, a API OpenMP foi utilizada com a função de prover o correto gerenciamento das GPUs de uma máquina, de modo que cada *thread* da CPU seja responsável por invocar um *kernel* CUDA em um específico *device*. O uso de múltiplas *threads* na CPU é necessário para reduzir o desequilíbrio causado quando uma simples *thread* invoca os *kernels* em todas as GPUs presentes. Se um número maior de GPUs estiver disponível, quando a única *thread* do *host* tiver terminado de iniciar o *kernel* na última GPU, provavelmente o primeiro *kernel* da primeira GPU avançou muito em seu processamento, ou até mesmo o terminou (Xavier, 2013).

3.2 MPI

O MPI (*Message Passing Interface*) é um padrão para troca de mensagens entre processos, que estabelece um padrão flexível e eficiente para o desenvolvimento de aplicações de alto desempenho. O padrão MPI permite criar aplicações paralelas de memória distribuída, já que os processos envolvidos na computação não precisam compartilhar suas memórias em um mesmo espaço de endereçamento (Pacheco et al., 1997). Existem diversas implementações deste padrão disponíveis, como o MPICH[1].

Na elaboração de uma aplicação MPI, é necessário identificar o número de processos que irão executar determinada tarefa. O ambiente MPI atribui a cada processo um *rank* e um grupo de comunicação. O *rank* é um identificador único de processo para cada tarefa MPI e os processos que fazem parte do mesmo grupo utilizam um mesmo comunicador para realizarem as trocas de mensagens. O identificador de cada processo dentro de um mesmo grupo pode ser obtido pelo uso da primitiva *MPI_Comm_rank*. As primitivas *MPI_Send* e *MPI_Recv*, ambas de natureza bloqueante, são consideradas ponto-a-ponto, já que apenas dois processos estarão envolvidos na comunicação: o processo A envia dados ao processo B por meio da chamada de *MPI_Send*, e B recebe os dados de A invocando a função *MPI_Recv* (Xavier, 2013).

Algumas funções podem envolver um grupo de dois ou mais processos. Pode-se citar, como exemplo, o *MPI_Allreduce*, que combina os valores de todos os processos e distribui o resultado de volta para todos, e o *MPI_Barrier*, utilizado para sincronização dos processos de um grupo, onde todos os processos deverão esperar até que todos alcancem a barreira para avançarem na execução (Pacheco et al., 2011; Xavier, 2013).

3.3 GPGPU

3.3.1 CUDA

A unidade de processamento gráfico, ou apenas GPU (*Graphics Processing Unit*), é uma placa de vídeo que possui grande poder de processamento, isso graças ao grande número de unidades de processamento disponíveis para processar dados simultaneamente. A ar-

[1] www.mpich.org/

quitetura da fabricante NVIDIA, CUDA (*Compute Unified Device Architecture*) é uma das plataformas mais populares para desenvolvimento de aplicações em GPUs. CUDA provê ferramentas de desenvolvimento e bibliotecas em linguagens como C, que conseguem esconder do programador a arquitetura interna das GPUs, facilitando assim o desenvolvimento de aplicações (Cook et al., 2012).

Para executar uma aplicação em CUDA, o programador deve criar uma função paralela chamada *kernel*. Um *kernel* é uma função que pode ser chamada na CPU mas que é executada simultaneamente por várias *threads* (que serão criadas em tempo de execução para processar os dados), onde cada *thread* é executada fisicamente em um dos muitos *stream processors* presentes na GPU. Assim, para aproveitar o máximo do desempenho oferecido pelo hardware, o programador deve subdividir seu problema em várias tarefas mais simples que possam ser executadas em paralelo por distintas *threads*.

As *threads* são agrupadas abstratamente em blocos de *threads*, ou apenas blocos. Cada bloco pode ser formado por grupos de *threads* organizadas abstratamente em uma, duas ou três dimensões. Assim como as *threads*, os blocos também podem ser agrupados em *grids*, que podem ser organizados em uma ou duas dimensões. Considerando que muitas *threads* executam o mesmo trecho de código, cada uma precisa de um identificador único. Esses identificadores serão utilizados para distinguir cada *thread* das demais, assim como também para determinar a porção de dados que cada uma irá processar. Para tal, a arquitetura CUDA provê mecanismos úteis para a obtenção destes identificadores: duas variáveis internas criadas em tempo de execução indicam de maneira única a posição de uma *thread* em um bloco (*threadIdx*) e a posição de um bloco de *threads* no *grid* (*blockIdx*) (Cook et al., 2012; Xavier, 2013; Rocha, 2012). A figura 3.1 representa o modelo da arquitetura CUDA.

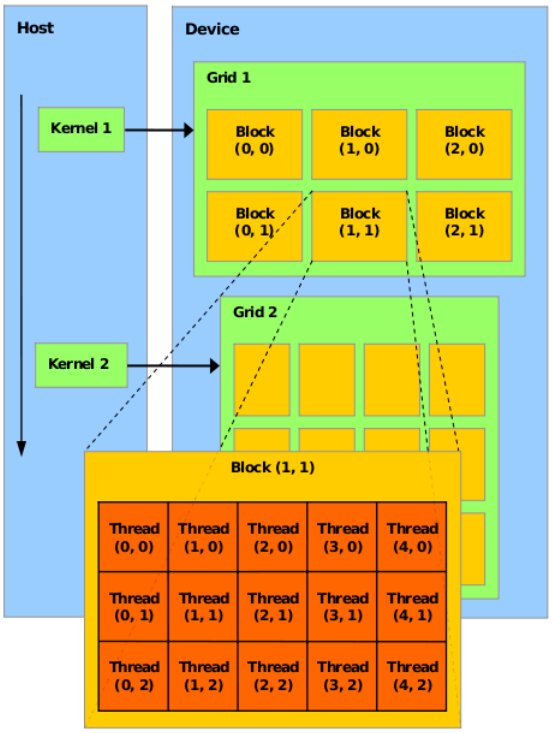

Figura 3.1: Arquitetura Cuda (CUDA C Programming Guide v6.5).

Em virtude da hierarquia de threads e de blocos, a memória em CUDA é organizada em três grupos distintos: memória local, memória compartilhada e memória global. A memória local, composta por registradores, é associada a cada *stream processor*, sendo portanto de acesso exclusivo à cada *thread*. A memória compartilhada pode ser acessada por todas as *threads* que pertençam ao mesmo bloco durante todo o ciclo de vida desse bloco. Já a memória global permite a leitura e escrita por qualquer *thread* que executa em qualquer bloco. Esta é a memória que pode ser acessada diretamente pela CPU. Do ponto de vista do tempo de acesso, a memória local possui o menor tempo de acesso, seguida pela memória compartilhada e por fim a memória global (Xavier, 2013; Rocha, 2012).

ECC

A forma exata de como o ECC (*Error Correction Check*) é implementado nas GPUs NVIDIA não está disponível publicamente. No geral memórias com ECC são comumente utilizadas nos sistemas modernos, especialmente em servidores e agregados computacionais, pois oferecem a capacidade de detecção e correção de erros oriundos do hardware.

Essa característica, que por padrão vem ativa, não somente detecta erros de bits simples, como também os corrige de forma transparente. ECC funciona como mostrado na figura 3.2. Em uma operação de escrita, o controlador da memória codifica os grupos ECC envolvidos usando algoritmos de codificação embutidos no dispositivo. O código do ECC (7 ou 8 bits) é armazenado com o dado na memória, consumindo, portanto, parte da memória para este fim. Na operação de leitura, os controladores de memória leem os grupos de ECC envolvidos, incluindo ambos, o dado e os códigos ECC. Esses códigos são recomputados com base nos dados lidos e comparados com os códigos ECC armazenados. Caso eles sejam diferentes, o controlador de memória automaticamente corrige os erros de bit simples e reporta erros de múltiplos bits usando uma interrupção, que é entregue ao sistema operacional. Isso permite um ganho em termos de confiabilidade, porém o desempenho é afetado com esse procedimento. Note que os dados utilizados em função do ECC reduzem o espaço total disponível para dados de usuários na GPU (por exemplo, 5.25 GB ao invés de 6GB, o que corresponde a 12.5% de redundância) (Qin et al., 2005; Habich et al., 2013).

Para avaliar o impacto relacionado ao se desligar o ECC, Freitas (2014), em sua dissertação de mestrado, analisa o tempo para se obter os resultados de sua simulação com e sem o ECC ativado. Com ECC ativo, o tempo de execução foi de aproximadamente 1773s; desativando-o o tempo foi reduzido para 1371s, ou seja, uma ganho de 29% no tempo de execução. Neste mesmo contexto, Habich et al. (2013) conclui que o uso do ECC causa uma perda entre 10% e 18% no desempenho da memória da GPU, causando uma queda entre 30% e 40% no desempenho total da aplicação.

Figura 3.2: Operação de Escrita\Leitura em memórias ECC (Qin et al., 2005).

3.3.2 UVA

O endereçamento virtual unificado, ou *Unified Virtual Addressing* (UVA), permite que a memória do *host* e de todos os *devices* instalados seja unificada em um único espaço virtual de endereçamento. Este recurso permite o estabelecimento de uma comunicação ponto-a-ponto entre duas GPUs, de modo que a leitura e escrita de dados em ambas seja efetuada de maneira direta, através do barramento PCI-e, sem a necessidade de passar pela memória do *host* (Xavier, 2013). Esta comunicação é apresentada na figura 3.3. Apesar de descrita nesta seção, essa característica não foi utilizada para solucionar o problema de comunicação entre a GPUs descrita na próxima seção. Neste trabalho foi empregada uma abordagem diferente para se lidar com este problema.

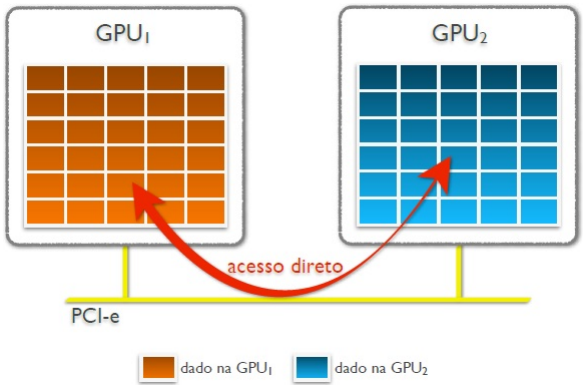

Figura 3.3: Comunicação utilizando UVA (Xavier, 2013).

3.4 Infiniband

Agregados de computadores são compostos por um conjunto de máquinas que necessitam muitas vezes se comunicar para trocar dados, o que é realizado através da rede de interconexão. Na maioria das vezes, o tempo necessário para que se realize esta comunicação é muito grande, o que pode acabar comprometendo o tempo total de execução das aplicações que executam neste tipo de ambiente. IBA (*Infiniband Architecture*) proporciona alta eficiência no gerenciamento de grandes volumes de informações e de equipamentos de uma rede, sendo projetada para fornecer baixa latência de comunicação e altas taxas de transferência de dados, podendo ultrapassar a relação de 10 *Gibabits* por segundo, limite imposto pela placas de padrão *Ethernet* (InfiniBand Trade Association et al., 2000).

A *Infiniband* é uma especificação de uma nova tecnologia de interconexão para redes compostas de nós processadores, dispositivos de entrada e saída de dados (E/S), chaveadores e roteadores. A figura 3.4 apresenta a organização de uma rede *Infiniband*. Uma rede IBA é dividida em sub-redes interconectadas por roteadores e, cada sub-rede compreende um ou mais chaveadores, nós processadores e dispositivos de E/S. Essa infraestrutura permite a troca de informações entre um nó processador e outro, entre nós processadores e dispositivos de E/S e entre diferentes sub-redes (Pfister et al., 2001).

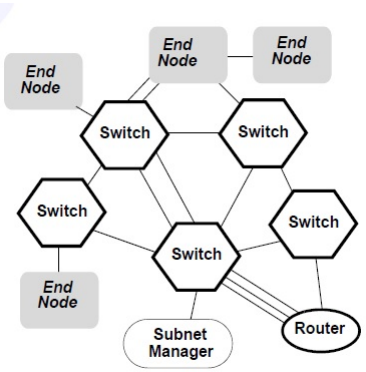

Figura 3.4: Organização de uma rede Infiniband (InfiniBand Trade Association et al., 2000).

A *Infiniband* possibilita a formação de uma rede de alta velocidade devido a forma como a comunicação é estruturada. Esta arquitetura também proporciona alta confiabilidade e tolerância a falhas na comunicação, uma vez que o hardware *Infiniband* é responsável por toda a troca de informações, sem necessitar dos recursos do processador, como acontece nas tecnologias tradicionais.

Com o intuito de avaliar a latência e a banda passante das arquiteturas de comunicação disponíveis (*Infiniband* e *Ethernet*), foram realizadas experimentos com tamanhos diferentes de pacotes sendo transportados por estas arquiteturas. Os gráficos a seguir mostram um comparativo com os resultados obtidos. Analisando os gráficos, fica evidente que a latência da *Infiniband* é muito inferior a da *Ethernet*, enquanto que a banda passante da *Infiniband* é aproximadamente 16 vezes superior à banda da *Ethernet*.

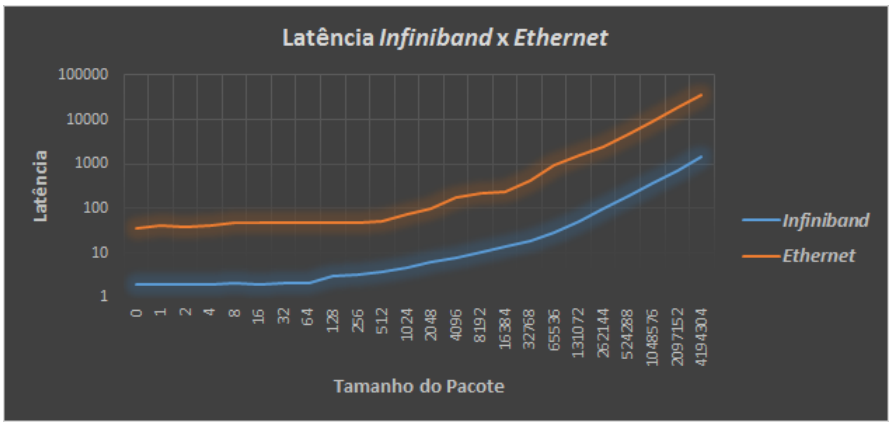

Figura 3.5: Latência da *Infiniband* e da *Ethernet*.

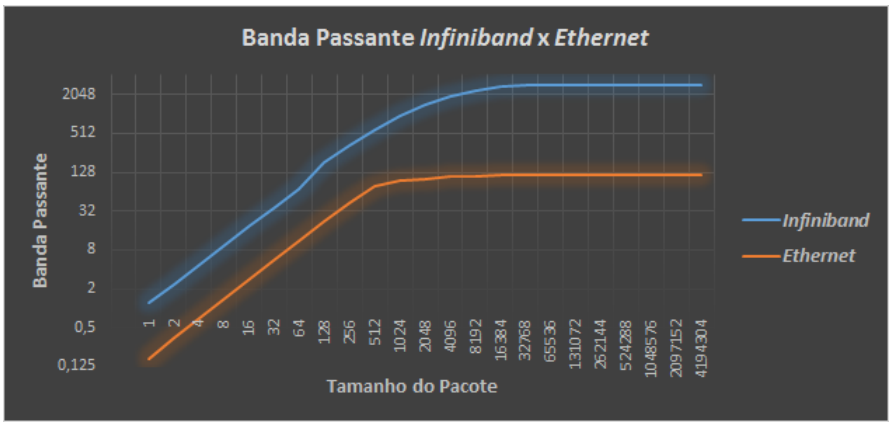

Figura 3.6: Banda passante da *Infiniband* e da *Ethernet*.

4 Implementação

A implementação do modelo computacional paralelo abordada neste trabalho utiliza como arcabouço a implementação desenvolvida por Xavier (2013), que faz uso de múltiplas GPUs localizadas em diferentes máquinas interligadas por uma rede. O código foi desenvolvido em C, empregando CUDA, OpenMP e MPI.

O espaço discretizado é dividido entre todas as GPUs, de maneira que cada uma irá operar em uma específica fatia do espaço original. A fração do tecido é dada pelo cálculo da divisão da dimensão x de uma malha (X, Y, Z) que descreve o tecido pelo número total de GPUs que participam da computação, N, restando uma malha $((X + N - 1)/N, Y, Z)$ a ser calculada em cada GPU (ou simplesmente *device*). Através do esquema de divisão adotado por Xavier (2013), o grupo de GPUs consegue processar todo o tecido.

Para se obter o número total de GPUs, N, que participa da computação foi utilizada a função *cudaGetDeviceCount*, que retorna o número de *devices* contidos em uma máquina. Cada processo MPI irá então enviar este valor a todos os demais processos do grupo. Esta operação é necessária porque cada GPU precisa definir a sua posição relativa em relação às demais GPUs, de modo a definir qual fatia do tecido deverá processar. A Figura 4.1 ilustra a estratégia de divisão para uma malha $(50, 20, 20)$ entre 5 GPUs.

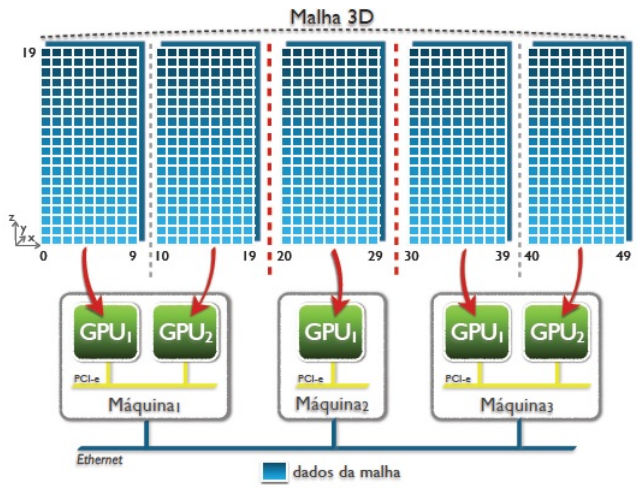

Figura 4.1: Exemplo de Divisão de um malha 3D (Xavier, 2013).

Xavier (2013) utiliza a API OpenMP para realizar o gerenciamento das GPUs de uma máquina, de modo que cada *thread* da CPU seja responsável por invocar um *kernel* CUDA em uma GPU específica da máquina. O uso de múltiplas *threads* se faz necessário para reduzir o desequilíbrio causada por uma única *thread* que lança vários *kernels* em GPUs diferentes, conforme explicado no capítulo anterior.

Na arquitetura CUDA, sabe-se que a configuração de execução exerce grande influência no desempenho da aplicação. Baseado nos recursos disponíveis do hardware da GPU, como também na demanda de memória por cada *thread* CUDA, o número de *threads* por bloco foi fixado neste trabalho em 256. O cálculo da dimensão do *grid* de blocos é realizado de forma automática, levando em conta as dimensões da fatia da malha e número de *threads* por bloco. Além disso, com o intuito de utilizar o maior número de núcleos possível (uma GPU possui oito vezes mais núcleos de processamento para lidar com cálculos de precisão simples que unidades de processamento para cálculos envolvendo precisão dupla), fato que contribui positivamente no desempenho da aplicação, foi utilizada a precisão simples em todos os cálculos do modelo. Contudo, tal escolha não resultou em prejuízos significativos que pusessem à prova a confiabilidade dos resultados numéricos.

Cada *thread* CUDA que executa o *kernel* representa um único ponto (x, y, z) no espaço discretizado do tecido tridimensional, sendo de sua responsabilidade o cálculo completo do conjunto de EDPs que representam as populações de células e moléculas simuladas no modelo. Durante o cálculo das EDPs, para que uma *thread* CUDA possa computar corretamente um ponto do tecido ela deve possuir acesso a dados produzidos pelas *threads* vizinhas. Com o auxílio de um *buffer*, uma *thread* que está realizando seus cálculos no tempo t consegue obter o acesso aos dados produzidos por seus vizinhos no tempo $t-1$. Assim, evita-se uma condição de corrida (ou *race condition*) entre as *threads*, visto que os dados novos produzidos por uma *thread* no tempo t somente são acessados por ela, pois seus vizinhos acessam os dados no tempo $t-1$. Essas duas entradas do *buffer* mudam seu papéis a cada passo de tempo para que uma cópia de dados seja evitada.

Devido ao esquema adotado de divisão da malha entre as GPUs, todos os dados vizinhos relacionados às dimensões y e z residem na própria GPU, restando apenas os

dados relacionados à dimensão x a serem devidamente tratados. Seja uma *thread* que representa um ponto no espaço 3D do tecido definida pelas coordenadas (α, β, γ). Em relação à dimensão x, um vizinho dessa *thread* pode ter as coordenadas (α - 1, β, γ) ou (α + 1, β, γ) e seu dado pode estar localizado em uma dentre 3 regiões distintas: a) na mesma GPU em que a *thread* realiza o cálculo; b) em uma GPU vizinha que resida na mesma CPU; ou c) em uma GPU que esteja em uma outra máquina. Nos dois últimos casos, os dados são chamados de bordas, pois se encontram nos extremos do eixo x de uma fatia de tecido processada por uma GPU. A seguir, a Figura 4.2 ilustra os três tipos de localização definidos anteriormente para um dado de um vizinho no eixo x.

Por meio de um identificador global, cada GPU consegue identificar o local onde deverá buscar o dado vizinho, ou na sua própria memória ou em outra GPU. Caso seja classificado como o primeiro caso - especificado em a) - o vizinho compartilha com a *thread* o mesmo espaço de memória, logo a *thread* poderá obter o dado requisitado com um simples acesso à memória global de sua GPU. Para as situações definidas em b) ou c), foram utilizados vetores auxiliares que irão armazenar os dados de borda da malha de seus vizinhos que estejam localizados em outras GPUs. Já que a cada passo de tempo os dados de borda são necessários em qualquer GPU para a correta computação dos pontos, tanto o envio quanto o recebimento de tais informações deverá ser realizado ao final de cada passo de tempo, ou seja, após o término da execução do *kernel* (Xavier, 2013).

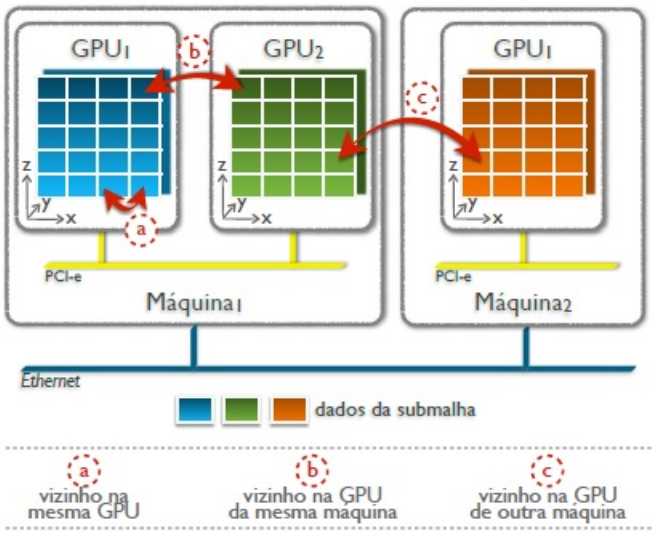

Figura 4.2: Localização de um dado vizinho de x (Xavier, 2013).

Para lidar com situações onde os dados de borda se encontram em endereços de memória de outras GPUs, há a necessidade de sincronização de todas as GPUs, de modo que as populações do modelo sejam calculadas corretamente a cada passo de tempo. Xavier (2013) implementa três diferentes abordagens para lidar com as trocas de borda. Na primeira abordagem os dados de borda são trocados entre duas GPUs que estejam ou não na mesma máquina utilizando a memória da CPU. A segunda abordagem faz uso de um recurso especial oferecido por GPUs da classe Fermi, ou em modelos mais atuais, o espaço virtual de endereçamento apresentado na seção 3.3.2. Na terceira estratégia foram criados dois *kernels* CUDA: um irá calcular os pontos de borda que serão utilizados na troca entre os processos e o outro será responsável por computar os pontos restantes, ou mais internos. A ideia é que por meio do uso de diferentes *streams* CUDA tal abordagem possa ser implementada, associando cada *kernel* a um diferente *stream*. Com o uso de dois *streams* CUDA, a tarefa de calcular o *kernel* para os pontos mais internos pode ser executada simultaneamente com a cópia dos pontos de borda para a memória da CPU, de modo que essa transferência de dados possa ser iniciada imediatamente após o cálculo da borda ter sido realizado. Ao utilizar tal estratégia, é esperado que se consiga sobrepor parte da comunicação com o processamento dos pontos mais internos da submalha, de modo a reduzir ainda mais o tempo de execução do modelo. Neste trabalho optou-se por implementar apenas a terceira abordagem proposta, onde a Figura 4.3 esquematiza a solução escolhida.

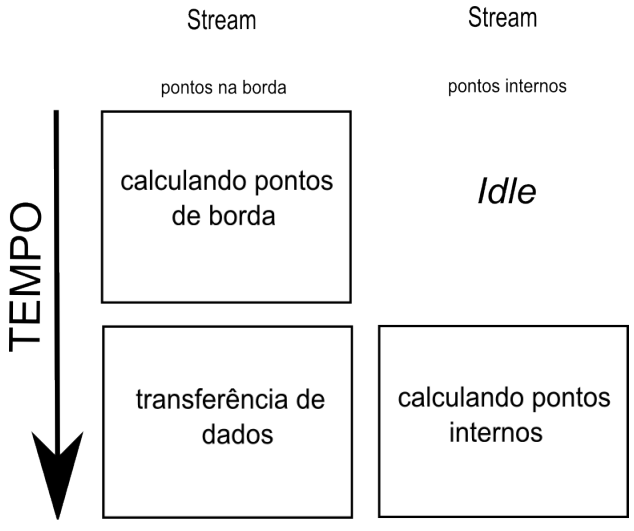

Figura 4.3: Operação utilizando *streams* CUDA.

5 Resultados

5.1 Experimentos Computacionais

Esta seção apresenta os resultados computacionais obtidos pelas simulação de 1 dia de infecção causada pela exposição de uma seção do tecido humano com 216 mm^3 de volume, a bactérias. O tecido é representado por uma malha de $60 \times 60 \times 60$ pontos. As figuras a seguir ilustram a evolução temporal da bactéria, dos neutrófilos e da citocina pró-inflamatória no tecido, 8 horas, 16 horas e 24 horas após o início da infecção. Para melhor visualização da simulação, foi feito um corte no plano z, evidenciando os resultados no interior do tecido.

Como condição inicial estabeleceu-se que a população de bactérias seria concentrada em um único ponto, enquanto que as populações das células e moléculas do modelo foi definida como nula em todo o tecido. Nas figuras 5.1, 5.2 e 5.3 é possível verificar que a bactéria não conseguiu uma difusão expressiva, devido à detecção e ao combate realizado pelos neutrófilos e macrófagos ao patógeno. Ao fim de 24 horas de simulação a população de neutrófilos está presente em praticamente todo o tecido (Figura 5.6), porém devido à função g (definida no Capítulo 2), as células de defesa do organismo são impedidas de alcançar a parte do tecido onde a concentração de bactérias é muito alta. Portanto, sua população nos pontos próximos onde a infecção começou (onde a presença de bactérias é mais elevada), não varia de forma significativa. A ação de substâncias como a citocina pró-inflamatória só se inicia após a detecção do antígeno no tecido. A citocina pró-inflamatória influencia no crescimento da população de macrófagos ativados. Como uma das fontes de produção da citocina é o neutrófilo, sua concentração fica próxima aos pontos onde os neutrófilos estão presentes (Figuras 5.7, 5.8 e 5.9). À medida que o antígeno vai sendo eliminado, a concentração de citocina anti-inflamatória vai aumentando, o que causa o bloqueio na ativação dos macrófagos ainda não ativados e na produção de citocinas pró-inflamatória.

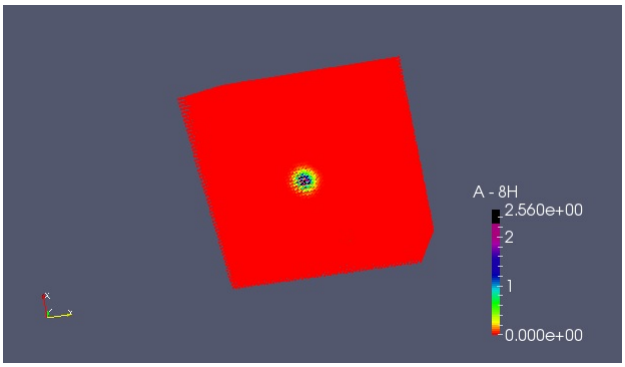

Figura 5.1: Bactérias após 8 horas de infecção.

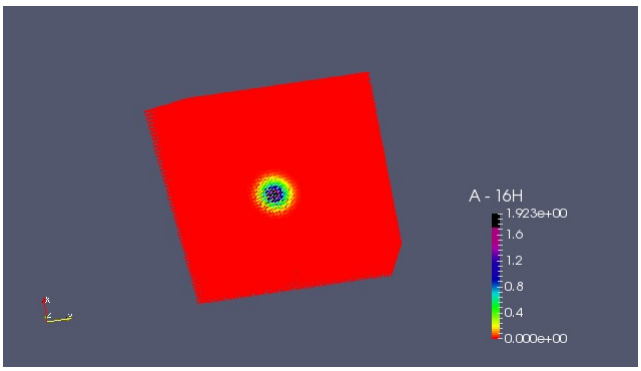

Figura 5.2: Bactérias após 16 horas de infecção.

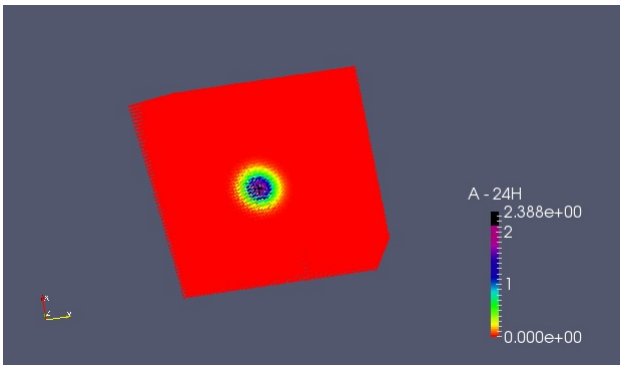

Figura 5.3: Bactérias após 24 horas de infecção.

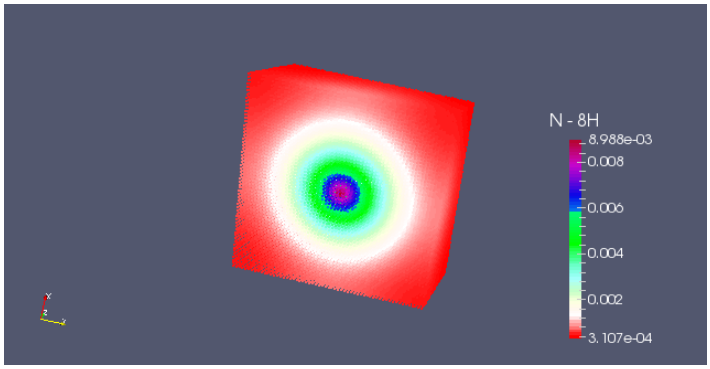

Figura 5.4: Neutrófilos após 8 horas de infecção.

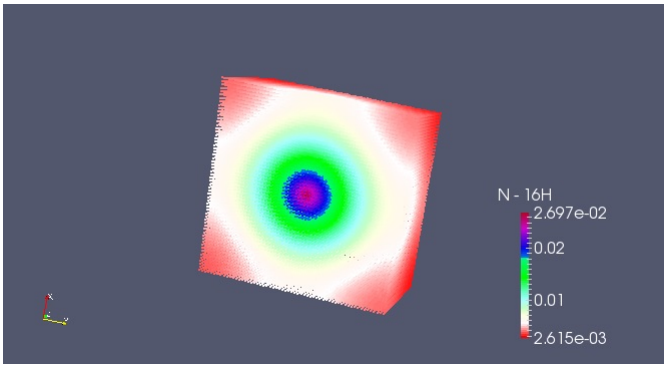

Figura 5.5: Neutrófilos após 16 horas de infecção.

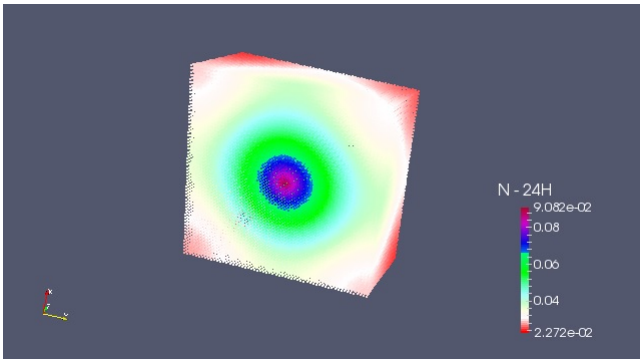

Figura 5.6: Neutrófilos após 24 horas de infecção.

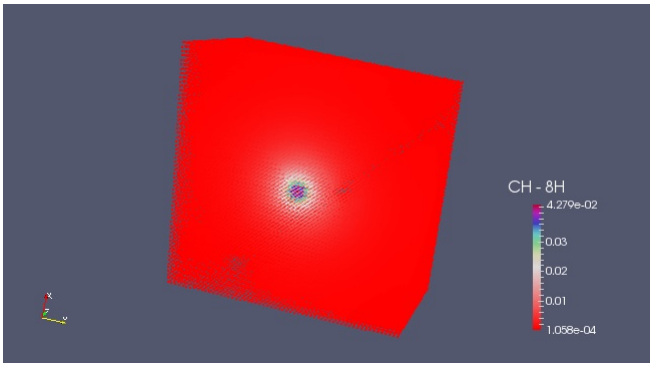

Figura 5.7: Citocina pró-inflamatória após 8 horas de infecção.

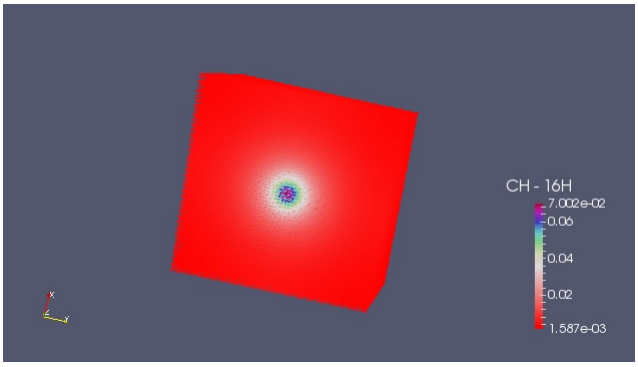

Figura 5.8: Citocina pró-inflamatória após 16 horas de infecção.

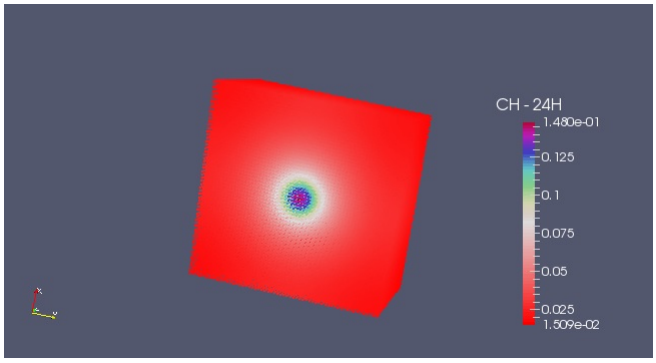

Figura 5.9: Citocina pró-inflamatória após 24 horas de infecção.

5.2 Arquitetura

Cada uma das máquinas utilizadas possui dois processadores AMD Opteron 6272 com 16 núcleos de 1400 MHz, 32 GB de memória RAM (*Random Access Memory*) e sistema operacional Linux 64-bits, distribuição Rocks 6.1.1, com kernel versão 2.6.32. As máquinas são interligadas através do cabo RJ45 com arquitetura de conexão *Gigabit Ethernet* e também são interligadas pela arquitetura de conexão *Infiniband*.

As simulações realizadas foram executadas em um conjunto de 2 e 4 máquinas, cada uma com duas GPUs NVIDIA Tesla M2075 com 448 núcleos CUDA cada. A GPU conta com 5.3GB de memória global, 65KB de memória para constantes, versão compute capability 2.0, e oferece suporte ao recurso UVA.

5.3 Parâmetros da Simulação

Com o intuito de avaliar o impacto no desempenho da aplicação, o modelo foi submetido a testes em diferentes cenários, resultantes de todas as combinações entre ECC habilitado e desabilitado, bem como com *Ethernet* e *Infiniband*. Os cenários são portanto: 1) *Infiniband* com ECC desligado; 2) *Infiniband* com ECC ligado; 3) *Ethernet* com ECC desligado e 4) *Ethernet* com ECC ligado.

Para cada experimento foram simulados diferentes tamanhos de tecido, com volumes de 125 mm^3, 1 cm^3, 3,375 cm^3, 8 cm^3 e 15,625 cm^3, representados pelas malhas de $50 \times 50 \times 50$, $100 \times 100 \times 100$, $150 \times 150 \times 150$, $200 \times 200 \times 200$ e $250 \times 250 \times 250$ pontos, respectivamente. Tais valores representam cenários válidos do ponto de vista biológico, uma vez que a simulação pode representar desde um pequeno pedaço de tecido a um órgão inteiro.

Uma outra variável do modelo é o tempo de simulação da infecção em número de dias. Cada dia de infecção é representado por 1.000.000 de iterações. Devido ao alto tempo gasto na simulação das malhas para a versão sequencial do código, apenas 1% do período equivalente a um dia de infecção (10.000 iterações) será simulado em todas as versões da aplicação e em todos os cenários. Segundo Xavier (2013), isso pode ser feito pois todo o processo realizado a cada iteração da simulação é dado sempre pelo mesmo

conjunto de instruções, cujo tempo de computação é muito próximo para qualquer valor usado como dado nas computações. Logo qualquer número de iterações pode ser tomado como uma amostra da simulação. Além disso, o maior foco deste trabalho consiste na avaliação das técnicas e abordagens empregadas para a redução no tempo total gasto em uma simulação e não em seus resultados biológicos propriamente ditos.

A versão sequencial do código utiliza somente a CPU para realizar as computações. Em todas as versões para múltiplas GPUs implementadas neste trabalho, o número de *threads* por bloco utilizado é igual a 256 e o tamanho do *grid* é calculado automaticamente com base nas dimensões da malha, no número total de GPUs participantes da computação e no tamanho do bloco de *threads*.

Cada configuração de dimensões da malha foi executada 5 vezes para garantir a confiabilidade dos tempos medidos: o desvio padrão ficou abaixo de $1,52\%$ para todos os cenários testados. A cada execução, a aplicação *time* do Linux foi utilizada para medir o tempo gasto na execução do programa.

O fator de aceleração, ou *speedup*, foi utilizado para mensurar os ganhos obtidos pelos códigos paralelos em relação ao código sequencial. O *speedup* foi ser calculado empregando a equação (5.1):

$$S(p) = \frac{t_s}{t_p}, \tag{5.1}$$

onde t_s é o tempo de execução sequencial e t_p é o tempo de execução da versão paralela.

5.4 Resultados Computacionais

As tabelas a seguir apresentam os ganhos obtidos executando o código em 2 máquinas com 1 GPU em cada uma (Tabela 5.1), 2 máquinas com 2 GPUs em cada uma (Tabela 5.2) e 4 máquinas com 2 GPUs cada uma (Tabela 5.3). Pode-se notar que os *speedups* obtidos para a primeira e a segunda configuração assumem uma maior aceleração para a malha $50 \times 50 \times 50$, e então a aceleração decresce até um determinado ponto, voltando a assumir valores crescentes para as malhas maiores. Já na terceira configuração o valor da aceleração para a malha $50 \times 50 \times 50$ é expressivamente menor do que as acelerações obtidas para as outras malhas. Nota-se também que ao se utilizar a *Infiniband* o tempo de execução, como esperado, decresce significativamente, principalmente nas malhas maiores, chamando a atenção para a terceira configuração que obteve um tempo de execução de $956,6$ vezes menor que o tempo da versão sequencial na malha $250 \times 250 \times 250$. Pode-se verificar também que para malhas maiores, o desligamento do ECC passa a se tornar mais evidente, e a medida que mais GPUs são empregadas no processamento da aplicação, maior sua influência em relação ao desempenho. Em especial, na terceira configuração apenas o desligamento do ECC foi responsável por aproximadamente 23% do *speedup* total alcançado. Os melhores *speedups* obtidos, em todas as três configurações, foram resultados da combinação da arquitetura *Infiniband* com o desligamento do ECC das GPUs, concretizando as expectativas deste trabalho.

Tabela 5.1: *Speedups* obtidos executando o código em 2 máquinas e 2 GPUs.

Malha	**Versão**	**Tempo médio (s)**	**Desvio Padrão**	***Seepdup***
50 x 50 x 50	Sequencial	5259,1424	-	-
	Inf - Ecc Desl	18,184	0,58%	**289,2**
	Inf - Ecc Lig	18,695	0,11%	281,3
	Eth - Ecc Desl	30,126	0,08%	174,5
	Eth - Ecc Lig	30,807	0,21%	170,7
100 x 100 x 100	Sequencial	42096,627	-	-
	Inf - Ecc Desl	160,171	0,03%	**262,8**
	Inf - Ecc Lig	166,279	0,01%	253,1
	Eth - Ecc Desl	206,004	0,34%	204,3
	Eth - Ecc Lig	210,36	0,25%	200,1
150 x 150 x 150	Sequencial	142848,555	-	-
	Inf - Ecc Desl	548,039	0,07%	**260,6**
	Inf - Ecc Lig	566,620	0,06%	252,1
	Eth - Ecc Desl	647,052	0,13%	220,7
	Eth - Ecc Lig	665,428	0,09%	214,6
200 x 200 x 200	Sequencial	334907,982	-	-
	Inf - Ecc Desl	1304,874	0,06%	**256,6**
	Inf - Ecc Lig	1342,158	0,04%	249,5
	Eth - Ecc Desl	1481,900	0,13%	225,9
	Eth - Ecc Lig	1517,687	0,06%	220,6
250 x 250 x 250	Sequencial	694455,572	-	-
	Inf - Ecc Desl	2576,452	0,16%	**269,5**
	Inf - Ecc Lig	2664,178	0,11%	260,6
	Eth - Ecc Desl	2855,242	0,09%	243,2
	Eth - Ecc Lig	2932,455	0,05%	236,8

Tabela 5.2: *Speedups* obtidos executando o código em 2 máquinas e 4 GPUs.

Malha	**Versão**	**Tempo médio (s)**	**Desvio Padrão**	***Seepdup***
50 x 50 x 50	Sequencial	5259,1424	-	-
	Inf - Ecc Desl	10,451	0,07%	**503,1**
	Inf - Ecc Lig	10,617	1,52%	495,3
	Eth - Ecc Desl	22,348	0,44%	235,3
	Eth - Ecc Lig	22,400	0,39%	234,7
100 x 100 x 100	Sequencial	42096,627	-	-
	Inf - Ecc Desl	84,720	0,80%	**496,8**
	Inf - Ecc Lig	86,336	0,85%	487,5
	Eth - Ecc Desl	129,369	0,64%	325,3
	Eth - Ecc Lig	131,605	0,60%	319,8
150 x 150 x 150	Sequencial	142848,555	-	-
	Inf - Ecc Desl	290,580	0,63%	**491,5**
	Inf - Ecc Lig	299,475	0,59%	476,9
	Eth - Ecc Desl	389,418	0,29%	366,8
	Eth - Ecc Lig	398,846	0,38%	358,1
200 x 200 x 200	Sequencial	334907,982	-	-
	Inf - Ecc Desl	670,858	0,61%	**499,2**
	Inf - Ecc Lig	687,802	0,37%	486,9
	Eth - Ecc Desl	842,678	0,05%	397,4
	Eth - Ecc Lig	865,956	0,04%	386,7
250 x 250 x 250	Sequencial	694455,572	-	-
	Inf - Ecc Desl	1351,199	0,04%	**513,9**
	Inf - Ecc Lig	1385,883	0,10%	501,0
	Eth - Ecc Desl	1606,449	0,30%	432,2
	Eth - Ecc Lig	1650,885	0,40%	420,6

Tabela 5.3: *Speedups* obtidos executando o código em 4 máquinas e 8 GPUs.

Malha	**Versão**	**Tempo médio (s)**	**Desvio Padrão**	***Seepdup***
50 x 50 x 50	Sequencial	5259,1424	-	-
	Inf - Ecc Desl	8,816	0,37%	**596,7**
	Inf - Ecc Lig	9,907	0,21%	530,8
	Eth - Ecc Desl	32,620	0,46%	161,2
	Eth - Ecc Lig	33,467	0,07%	157,1
100 x 100 x 100	Sequencial	42096,627	-	-
	Inf - Ecc Desl	52,444	0,58%	**802,6**
	Inf - Ecc Lig	56,503	0,20%	745,0
	Eth - Ecc Desl	142,726	0,10%	294,9
	Eth - Ecc Lig	145,373	0,11%	289,5
150 x 150 x 150	Sequencial	142848,555	-	-
	Inf - Ecc Desl	167,777	0,10%	**851,4**
	Inf - Ecc Lig	171,495	0,63%	832,9
	Eth - Ecc Desl	364,703	0,03%	391,6
	Eth - Ecc Lig	365,508	0,21%	390,8
200 x 200 x 200	Sequencial	334907,982	-	-
	Inf - Ecc Desl	361,792	0,06%	**925,6**
	Inf - Ecc Lig	371,921	0,25%	900,4
	Eth - Ecc Desl	716,434	0,05%	467,4
	Eth - Ecc Lig	724,356	0,11%	462,3
250 x 250 x 250	Sequencial	694455,572	-	-
	Inf - Ecc Desl	725,948	0,05%	**956,6**
	Inf - Ecc Lig	763,524	0,24%	909,9
	Eth - Ecc Desl	1261,489	0,02%	550,5
	Eth - Ecc Lig	1293,770	0,33%	536,7

6 Conclusão

Este trabalho estendeu o modelo unidimensional originalmente proposto por Pigozzo et al. (2013) para um domínio tridimensional, bem como paralelizou o modelo resultante utilizando como arcabouço o modelo computacional proposto por Xavier (2013). A implementação paralela resultante fez uso de múltiplas GPUs em um agregado computacional para reduzir os tempos de computação.

Foi então analisado o impacto no desempenho ao se utilizar *Infiniband*, uma arquitetura de rede específica para realizar computação que requer baixa latência e alta banda passante. Verificou-se que o emprego da *Infiniband* contribuiu de forma significativa para a redução do tempo necessário para a execução do modelo: a conexão *Infiniband* ficou responsável pela maior parte do ganho conquistado no desempenho em relação aos testes que utilizou-se a comunicação *Ethernet*. Em relação ao *Error Correction Check* (ECC), nota-se que quanto mais acesso à memória necessário na execução da aplicação mais influência ele exerce no desempenho. Verificou-se que aplicações que não precisam de um processamento expressivo, o ECC pouco influencia. Ao combinar estes dois parâmetros, em aplicações que são executadas em agregados computacionais, pode-se obter um ganho substancial.

Novas técnicas podem ser empregadas para se reduzir o custo de execução das aplicações, como por exemplo estratégias de divisão de trabalho, onde o balanceamento de carga entre CPU e GPU permite que as malhas podem também ser calculadas pelos processadores disponíveis na CPU e não apenas pelas GPUs. Um outro recurso que pode ser explorado em relação a arquitetura do *hardware*, é a utilização de uma parte da memória RAM das GPUs como memória *cache*, que pode ser acessada mais rapidamente se comparada a outro tipo de memória.

Referências Bibliográficas

Byrne, H. M.; Owen, M. R. A new interpretation of the Keller-Segel model based on multiphase modelling. **Journal of mathematical biology**, v.49, n.6, p. 604–626, 2004.

Chandra, R. **Parallel Programming in OpenMP**. High performance computing. Morgan Kaufmann Publishers, 2001.

Cook, S. **CUDA Programming: A Developer's Guide to Parallel Computing with GPUs**. Applications of GPU Computing Series. Morgan Kaufmann, 2012.

Reis, R. F. **Simulações numéricas 3D em ambiente paralelo de hipertemia com nanopartículas magnéticas**. Juiz de Fora, 2014. Dissertação de Mestrado - Universidade Federal de Juiz de Fora.

Goutelle, S.; Maurin, M.; Rougier, F.; Barbaut, X.; Bourguignon, L.; Ducher, M. ; Maire, P. The hill equation: a review of its capabilities in pharmacological modelling. **Fundamental & clinical pharmacology**, v.22, n.6, p. 633–648, 2008.

Habich, J.; Feichtinger, C.; Kostler, H.; Hager, G. ; Wellein, G. Performance engineering for the lattice Boltzmann method on GPGPUs: Architectural requirements and performance results. **Computers & Fluids**, v.80, n.0, p. 276 – 282, 2013. Selected contributions of the 23rd International Conference on Parallel Fluid Dynamics ParCFD2011.

Association, I. T.; others. **InfiniBand Architecture Specification: Release 1.0**. InfiniBand Trade Association, 2000.

LeVeque, R. **Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time-Dependent Problems**. Society for Industrial and Applied Mathematics, 2007.

Pacheco, P. **Parallel Programming with MPI**. Morgan Kaufmann Publishers, 1997.

Pacheco, P. **An Introduction to Parallel Programming**. An Introduction to Parallel Programming. Morgan Kaufmann, 2011.

Painter, K. J.; Hillen, T. Volume-filling and quorum-sensing in models for chemosensitive movement. **Can. Appl. Math. Quart**, v.10, n.4, p. 501–543, 2002.

Painter, K. J.; Sherratt, J. A. Modelling the movement of interacting cell populations. **Journal of theoretical biology**, v.225, n.3, p. 327–339, 2003.

Painter, K. J. Continuous models for cell migration in tissues and applications to cell sorting via differential chemotaxis. **Bulletin of mathematical biology**, v.71, n.5, p. 1117–1147, 2009.

Pfister, G. F. An introduction to the infiniband architecture. **High Performance Mass Storage and Parallel I/O**, v.42, p. 617–632, 2001.

Pigozzo, A. B. **Implementação computacional de um modelo matemático do sistema imune inato**. Juiz de Fora, 2011. Dissertação de Mestrado - Universidade Federal de Juiz de Fora.

Pigozzo, A. B.; Macedo, G. C.; Weber dos Santos, R. ; Lobosco, M. Computational modeling of microabscess formation. **Computational and mathematical methods in medicine**, v.2012, 2012.

Pigozzo, A. B.; Macedo, G. C.; dos Santos, R. W. ; Lobosco, M. On the computational modeling of the innate immune system. **BMC Bioinformatics**, v.14, n.Suppl 6, p. S7, 2013.

Qin, F.; Lu, S. ; Zhou, Y. **Safemem: Exploiting ecc-memory for detecting memory leaks and memory corruption during production runs**. In: High-Performance Computer Architecture, 2005. HPCA-11. 11th International Symposium on, p. 291–302. IEEE, 2005.

Rocha, P. A. F. **Emprego de GPGPUs para acelerar simulações do sistema imune inato**. Juiz de Fora, 2012. Dissertação de Mestrado - Universidade Federal de Juiz de Fora.

Rocha, P.; Xavier, M.; Pigozzo, A.; de M. Quintela, B.; Macedo, G.; dos Santos, R. ; Lobosco, M. **A three-dimensional computational model of the innate immune system**. In: Murgante, B.; Gervasi, O.; Misra, S.; Nedjah, N.; Rocha, A.; Taniar, D. ; Apduhan, B., editors, Computational Science and Its Applications - ICCSA 2012, volume 7333 de **Lecture Notes in Computer Science**, p. 691–706. Springer Berlin Heidelberg, 2012.

Roe, P.; Chattot, J. **Innovative Methods for Numerical Solutions of Partial Differential Equations**. World Scientific, 2002.

Wang, Z. On chemotaxis models with cell population interactions. **Mathematical Modelling of Natural Phenomena**, v.5, n.03, p. 173–190, 2010.

Xavier, M. P. **Implementação paralela em um ambiente de múltiplas gpus de um modelo 3d do sistema imune inato**. Juiz de Fora, 2013. Dissertação de Mestrado - Universidade Federal de Juiz de Fora.

Xavier, M.; do Nascimento, T.; dos Santos, R. ; Lobosco, M. **Use of multiple gpus to speedup the execution of a three-dimensional computational model of the innate immune system**. In: Journal of Physics: Conference Series, volume 490, p. 012075. IOP Publishing, 2014.