Simulações computacionais de escoamento de fluidos e problemas de reação-difusão através do método de Lattice Boltzmann

Joventino de Oliveira Campos

JUIZ DE FORA

AGOSTO, 2013

Simulações computacionais de escoamento de fluidos e problemas de reação-difusão através do método de Lattice Boltzmann

Joventino de Oliveira Campos

Universidade Federal de Juiz de Fora

Instituto de Ciências Exatas

Departamento de Ciência da Computação

Bacharelado em Ciência da Computação

Orientador: Bernardo Martins Rocha

JUIZ DE FORA

AGOSTO, 2013

Simulações computacionais de escoamento de fluidos e problemas de reação-difusão através do método de Lattice Boltzmann

Joventino de Oliveira Campos

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:

Bernardo Martins Rocha
Mestre em Modelagem Computacional

Rodrigo Weber dos Santos
Doutor em Matemática

Luis Paulo da Silva Barra
Doutor em Engenharia Civil

JUIZ DE FORA

23 DE AGOSTO, 2013

Aos meus amigos e à minha irmã.

Aos pais, pelo apoio e sustento.

Resumo

Recentemente a modelagem computacional vem sendo utilizada para o entendimento de fenômenos complexos nas mais diversas áreas. A partir dos princípios físicos, matemáticos e do conhecimento sobre o problema, chega-se a um modelo matemático, que descreve o fenômeno de interesse. A solução exata do modelo pode ser muito difícil de ser encontrada ou pode não existir. Então são usados métodos numéricos, para a resolução destas equações, como acontece em diversas aplicações em engenharia, biologia e física. A solução numérica da maioria dos modelos é extremamente custosa devido à alta resolução espacial e temporal exigida. Em geral, os métodos numéricos mais utilizados para a solução destes problemas são o método dos elementos finitos (MEF) e o método dos volumes finitos (MVF). Uma alternativa é o uso do método de Lattice Boltzmann (MLB), o qual tem sido cada vez mais utilizado para simulação de problemas complexos de dinâmica dos fluidos. O objetivo deste trabalho é apresentar a aplicação do método de Lattice Boltzmann aos problemas de dinâmica dos fluidos e atividade elétrica do coração, assim como avaliar o seu desempenho em ambientes de computação paralela recentes.

Palavras-chave: simulação de fluidos, reação-difusão, método de Lattice Boltzmann, eletrofisiologia cardíaca.

Abstract

Computational modeling has been used for understanding complex phenomena in various areas recently. A mathematical model can be obtained with the knowledge from the physical and mathematical principles of the problem, which describes the phenomenon of interest. The model exact solution can be very difficult to find or may not exist. Then numerical methods are used for solving these equations in many engineering, biology and physics applications. The models numerical solution is extremely costly due to the high spatial and temporal resolution required. The numerical methods most commonly used to solve these problems are the finite element method (FEM) and finite volume method (FVM). An alternative is the Lattice Boltzmann Method (LBM), which have been increasingly used for simulation of complex problems in fluid dynamics. The objective of this work is to present the application of the Lattice Boltzmann method to problems of fluid dynamics and electrical activity of the heart as well as evaluate its performance in recent parallel computing environments.

Keywords: fluid simulation, reaction-diffusion, lattice Boltzmann method, cardiac electrophysiology.

Agradecimentos

À minha irmã e demais parentes, pelo encorajamento e apoio.

Aos amigos de turma, que sempre estavam unidos ajudando uns aos outros.

Ao professor Bernardo Martins Rocha pela orientação, amizade e principalmente, pela paciência, sem a qual este trabalho não se realizaria.

Ao Grupo de Educação Tutorial da Engenharia Computacional, que muito contribuiu para a minha formação acadêmica.

Aos professores dos Departamentos de Ciência da Computação e Mecânica Aplicada e Computacional pelos seus ensinamentos e aos funcionários do curso, que durante esses anos, contribuíram de algum modo para o nosso enriquecimento pessoal e profissional.

"Everything should be made as simple as possible, but not simpler."

Albert Einstein

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviações

CPU	Unidade Central de Processamento
DFC	Dinâmica dos Fluidos Computacional
EDO	Equação Diferencial Ordinária
EDP	Equação Diferencial Parcial
GPU	Unidade de Processamento Gráfico
LGAC	Lattice Gas Autômato Celular
LR	Luo-Rudy
MDF	Método das Diferenças Finitas
MEF	Método dos Elementos Finitos
MLB	Método de Lattice Boltzmann
MPI	Message Passing Interface
MS	Mitchell-Schaeffer
MVF	Método dos Volumes Finitos

1 Introdução

Recentemente a modelagem computacional vem sendo utilizada para o entendimento de fenômenos complexos nas mais diversas áreas. A partir dos princípios físicos, matemáticos e do conhecimento sobre o problema, chega-se a um modelo matemático, que descreve o fenômeno de interesse. A solução exata do modelo pode ser muito difícil de ser encontrada ou pode não existir. Então são usados métodos numéricos, para a resolução destas equações, como acontece em diversas aplicações em engenharia, biologia e física.

1.1 Motivação

Fluidos estão presentes na água, no ar e até no corpo humano e seu movimento tem sido estudado há muitos anos. Problemas de engenharia podem precisar levar em conta a dinâmica dos fluidos, como por exemplo, a interação entre o vento e a estrutura de uma ponte. Os fluidos são muito importantes nos problemas de aerodinâmica, como na construção de aviões e carros de corrida. No começo os estudos sobre fluidos eram feitos com a análise matemática teórica e com experimentos, como o túnel de vento. Entretanto a resolução matemática pode ser muito difícil ou impraticável e a realização de experimentos pode ser muito cara. Uma alternativa a estes métodos é o uso de simulações computacionais para estudar fenômenos complexos como o escoamento de fluidos.

O movimento de um fluido pode ser descrito por um modelo matemático baseado em um conjunto de equações diferencias parciais (EDPs), conhecidas como equações de Navier-Stokes. Estas equações só possuem solução analítica para alguns casos simples. Sendo assim, o uso de métodos numéricos para se encontrar soluções aproximadas se faz necessário. A dinâmica dos fluidos computacional (DFC) é uma área da computação científica que utiliza métodos numéricos para a resolução das equações que descrevem o comportamento de fluidos.

Outros problemas de grande interesse de pesquisa podem ser modelados por equações diferencias parciais do tipo reação-difusão, como a modelagem do crescimento de

tumores, a simulação da sinapse do cérebro ou a modelagem da eletrofisiologia cardíaca. Este tipo de equação surgiu da modelagem de processos químicos, onde a concentração de uma substância se dispersa pelo domínio ao mesmo tempo que reage com outras substâncias. A equação possui um termo de difusão, como na equação do calor e possui um termo reativo, que pode ser uma determinada função ou sistema de equações diferencias ordinárias.

Existem diversos métodos numéricos que podem ser utilizados para a resolução destes modelos, alguns exemplos são o método das diferenças finitas (MDF), método dos elementos finitos (MEF) e método dos volumes finitos (MVF). Tais métodos se baseiam na discretização das equações que descrevem o movimento do fluido macroscopicamente e os sistemas resultantes são resolvidos usando métodos numéricos tradicionais tais como decomposição LU ou métodos iterativos como o método do Gradiente Conjugado.

Outra classe de métodos utilizados para simulação computacional de fluidos envolve os métodos *Lattice-Gas* Autômato Celular (LGAC) e o Método de Lattice Boltzmann (MLB). Tais métodos partem das propriedades microscópicas da dinâmica dos fluidos, para descrever o fenômeno na escala macroscópica. É possível mostrar matematicamente através de uma análise multi-escala que o tanto LGAC quanto o MLB, sob certas circunstâncias, são capazes de recuperar o comportamento macroscópico das equações de Navier-Stokes e portanto tem sido muito utilizados para a simulação de fluidos. Além disso, o MLB também pode ser adaptado para a solução numérica de problemas de reação-difusão.

Um problema comum em diversas aplicações de interesse prático que utilizam simulações computacionais é que muitas vezes essas simulações demandam um grande esforço computacional. Em muitas situações práticas, simulações podem demorar horas ou dias para executar, ou podem até mesmo serem tão grandes em termos de dados que a memória de apenas um computador é insuficiente. Um meio de reduzir este tempo de execução ou tornar a simulação viável é a utilização de computação paralela, que consiste em dividir o problema em partes menores a serem executadas concorrentemente, por exemplo, em diferentes processadores. Em geral esse tipo de abordagem traz uma grande redução no tempo de execução, permitindo que os problemas possam ser resolvidos rapidamente ou que problemas antes intratáveis, possam ser resolvidos. Dentro desse

contexto, o MLB é bem atrativo, pois é um algoritmo que pode ser paralelizado facilmente, permitindo obter um grande ganho em eficiência computacional. Neste trabalho o MLB será paralelizado utilizando duas técnicas de computação paralela, a primeira técnica é a paralelização por passagem de mensagem, utilizando a biblioteca MPI e a segunda, utilizando a plataforma CUDA, que é executada em unidades de processamento gráfico.

1.2 Justificativa

Considerando que grande parte das operações que o MLB realiza são independentes e podem ser feitas em paralelo, neste trabalho será feita a implementação dos métodos LGAC e MBL, utilizando MPI e CUDA.

MPI é um padrão de comunicação para programação paralela, onde o código é dividido em processos que podem se comunicar através da passagem de mensagem. Os processos podem executar em máquinas diferentes e em qualquer plataforma desde que o padrão MPI seja compatível.

CUDA é uma plataforma de computação paralela e um modelo de programação desenvolvido pela NVIDIA. Ela aproveita o potencial da unidade de processamento gráfico (GPU), para aumentar significativamente o desempenho de uma determinada aplicação.

Os métodos descritos podem ser utilizados em aplicações complexas da dinâmica dos fluidos, como meios porosos, escoamentos multifásicos e escoamentos turbulentos. Apesar do método MLB ser utilizado para problemas da dinâmica dos fluidos, ele também pode ser usado para outros tipos de problemas, como problemas de calor, difusão, e problemas como a eletrofisiologia cardíaca, reação-difusão, que são fenômenos de grande interesse de pesquisa. E quanto mais rápido estes problemas puderem ser resolvidos, mais avanços e estudos poderão ser feitos dentro da dinâmica dos fluidos e em outros campos. Além disso, a simulação em tempo real de problemas da eletrofisiologia pode ajudar no tratamento de pacientes. Sendo assim, este trabalho visa estudar e implementar os métodos LGAC e MLB para simulação de fluidos e eletrofisiologia cardíaca, além de otimizá-los através de programação paralela, para que as simulações sejam eficientes e rápidas. Para a verificação e validação da implementação do método MLB, serão realizadas aplicações com o código desenvolvido, comparando os resultados obtidos com a solução analítica

de problemas modelos, ou ainda comparar os resultados deste trabalho com resultados experimentais de trabalhos já validados.

O principal objetivo deste trabalho é estudar, analisar e implementar o método MLB, para simulações de fenômenos físicos complexos como o escoamento de fluidos e problemas de reação-difusão. São métodos recentes que vem se mostrando atrativos para a simulação da dinâmica de fluidos desde escoamentos externos como o vento ao redor da estrutura de um avião até a simulação da hemodinâmica de aneurismas (Golbert et al, 2009).

O presente trabalho é voltado a aplicação e implementação dos métodos descritos. Será feita a implementação do MLB e sua análise numérica, através de testes com problemas clássicos da dinâmica dos fluidos, já conhecidos como *benchmarks* para métodos de dinâmica dos fluidos computacional. O MLB também será aplicado ao problema de reação-difusão da eletrofisiologia cardíaca. Na sequência, devido ao alto custo computacional, procura-se otimizar a implementação, através do uso de computação paralela e avaliar seu desempenho neste ambiente. O método MLB possui grande potencial para computação paralela, devido ao grande número de operações que podem ser feitas simultaneamente. Então, busca-se estudar uma biblioteca do padrão *Message Passing Interface* (MPI) e a plataforma CUDA, a primeira voltada para paralelização executada por processadores e a segunda voltada para paralelização utilizando a unidade de processamento gráfico. Após o estudo procura-se implementar o método MLB nos paradigmas de computação paralela apresentados acima.

1.3 Organização da Monografia

O segundo capítulo trata dos aspectos teóricos e práticos do método Lattice-Gas Autômato celular, onde são apresentados os modelos que podem ser utilizados e como podem ser implementados. O terceiro capítulo aborda o método de Lattice Boltzmann para problemas de dinâmica dos fluidos, partindo da derivação de sua equação, apresentando modelos utilizados, e chegando então até os detalhes de implementação. O quarto capítulo descreve o método de Lattice Boltzmann, voltado a problemas do tipo reação-difusão e em particular descreve uma aplicação biológica envolvendo a modelagem da atividade

elétrica do tecido cardíaco usando o MLB. O quinto capítulo apresenta de forma sucinta as técnicas de computação paralela utilizadas neste trabalho, assim como detalhes de implementação específicos do MLB. O sexto capítulo apresenta os resultados obtidos com os métodos LGAC e MLB, para os problemas escoamentos de fluidos e para o modelo de eletrofisiologia cardíaca, além de apresentar os resultados com as implementações paralelas. O último capítulo apresenta as conclusões alcançadas com este trabalho e descreve algumas ideias de trabalhos futuros.

2 Lattice-Gas Autômato Celular

Neste capítulo será apresentado uma breve definição de Autômato Celular e em seguida será apresentado um autômato celular utilizado para a simulação de dinâmica dos fluidos, chamado Lattice-Gas Autômato Celular (LGAC).

2.1 Autômato Celular

O método LGAC é um autômato celular usado para a simulação de fluidos. Segundo (Wolf-Gladrow et al, 2000) um autômato celular é uma malha de células do mesmo tipo, onde cada célula possui um estado k, onde k é um número inteiro. O estado de todas as células é atualizado a cada passo de tempo de acordo com um conjunto de regras pré-estabelecidas levando em consideração o estado das células vizinhas.
Um exemplo de autômato celular é o Jogo da Vida, que descreve a evolução de uma população de indivíduos. Cada célula pode ter dois estados: vivo ou morto. A regra desse autômato celular pode ser descrita da seguinte forma:

- Uma determinada célula viva continuará viva, no próximo passo de tempo, se ela possuir dois ou três vizinhos vivos. Caso contrário ela morre.
- E uma célula morta pode se tornar viva, no próximo passo de tempo, se ela tiver três vizinhos vivos.

Autômatos celulares tem sido aplicados a diferentes problemas, como processos biológicos, terremotos, formação de galáxias e na dinâmicas dos fluidos através do LGAC (Wolf-Gladrow et al, 2000).

2.2 Lattice Gas Autômato Celular

O método *Lattice Gas* Autômato Celular (LGAC) é um autômato celular com algumas modificações e pode ser usado para a simulação de escoamento de fluidos. O LGAC busca

reproduzir a dinâmica dos fluidos através de um conjunto de partículas, todas com a mesma massa e velocidade, em módulo, situadas em um *lattice* regular. As velocidades dessas partículas são limitadas a um conjunto finito de direções, o qual depende do tipo de *lattice*.

Em cada ponto do *lattice* pode ou não existir partículas se deslocando com sua velocidade em uma determinada direção. Enquanto no autômato celular temos apenas um conjunto de regras de atualização, no LGAC estas regras são divididas em duas partes: colisão e propagação. A etapa de colisão é semelhante as regras de atualização do autômato celular, onde o estado de uma partícula pode ser alterado devido à uma colisão frontal de duas partículas, por exemplo. Na etapa de propagação, as partículas se deslocam na malha, de acordo com a sua direção de velocidade.

A dinâmica do método consiste na inicialização do estado de cada partícula na malha. Em seguida, para cada célula as etapas de colisão e propagação são realizadas. No método LGAC utiliza-se um conjunto de variáveis booleanas $n_i(\mathbf{x}, t), i = 0, ...l-1$, onde l é o número de direções de velocidade que a partícula pode assumir, indicando se existe $(n_i = 1)$ ou não existe $(n_i = 0)$ partícula na posição $\mathbf{x}$, no instante de tempo t se movendo na direção dada por $\mathbf{e}_i$. A equação que descreve a dinâmica do método é definida a seguir:

$$n_i(\mathbf{x} + \mathbf{e}_i, t+1) = n_i(\mathbf{x}, t) + \Delta_i(n_i(\mathbf{x}, t)), \;\; i = 0, ..., l-1 \;\; , \tag{2.1}$$

onde Δ é o operador de colisão (Golbert et al, 2009) e a complexidade deste operador cresce exponencialmente com o aumento do número de direções que as partículas podem assumir. De acordo com a equação, em um determinado passo de tempo, a partícula i de uma célula na posição $\mathbf{x}+\mathbf{e}_i$ recebe a partícula i da posição $\mathbf{x}$ do passo de tempo anterior, que pode ter sofrido mudança na sua direção devido ao operador Δ.

2.2.1 Modelo HPP

O primeiro LGAC foi o HPP (Hardy, Pazzis, Pomeau), ele é um autômato celular com *lattice* bidimensional, onde cada nó possui quatro partículas com massa unitária e uma das velocidades discretas. A Figura 2.1 mostra um exemplo de *lattice* para este modelo

de LGAC. O HPP possui 16 diferentes estados para um nó de seu *lattice* e os possíveis valores para a velocidade de uma partícula são:

$$\mathbf{e}_0 = (1,0);\ \ \mathbf{e}_1 = (0,1);\ \ \mathbf{e}_2 = (-1,0);\ \ \mathbf{e}_3 = (0,-1) \tag{2.2}$$

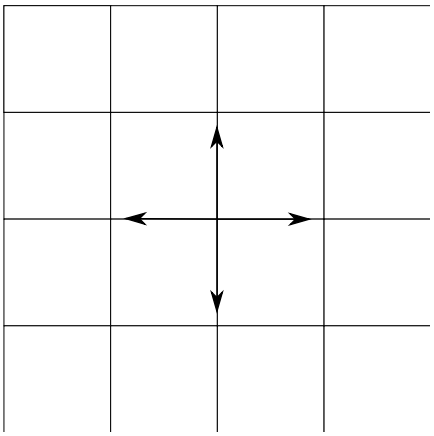


Figura 2.1: Lattice para o modelo HPP.

Então, a cada passo de tempo Δt, as partículas de um nó podem se colidir, alterar sua direção e em seguida se propagar para um dos nós vizinhos. A etapa de propagação pode ser expressa pela seguinte equação:

$$n_i(\mathbf{x} + \mathbf{e}_i, t+1) = n_i(\mathbf{x}, t),\ \ i = 0, ..., l-1 \tag{2.3}$$

A colisão de partículas só acontecerá quando um nó tiver duas partículas com velocidade na mesma direção e sentido contrário. Além disso a velocidade das partículas na outra direção deve ser zero, ou seja, só acontece colisão frontal e quando esta acontece as partículas envolvidas sofrem mudança na sua direção. O operador de colisão do HPP pode ser expresso como:

$$\Delta_i(\mathbf{n}) = -n_i\overline{n}_{i+1}n_{i+2}\overline{n}_{i+3} + \overline{n}_i n_{i+1}\overline{n}_{i+2}n_{i+3} \tag{2.4}$$

onde o índice de n é cíclico e varia de 0 a 3, $\overline{n}_i = 1 - n_i$ e Δ_i pode ser assumir três valores: $\Delta_i = -1$ aconteceu a colisão e as partículas saíram da direção atual, $\Delta_i = 0$ não aconteceu colisão e $\Delta_i = 1$ após a colisão uma direção que estava vazia recebe uma partícula. Juntando as Equações 2.3 e 2.4 obtêm-se a Equação da dinâmica do HPP, que

é dada pela equação 2.1. A Figura 2.2 exemplifica a dinâmica do modelo HPP.

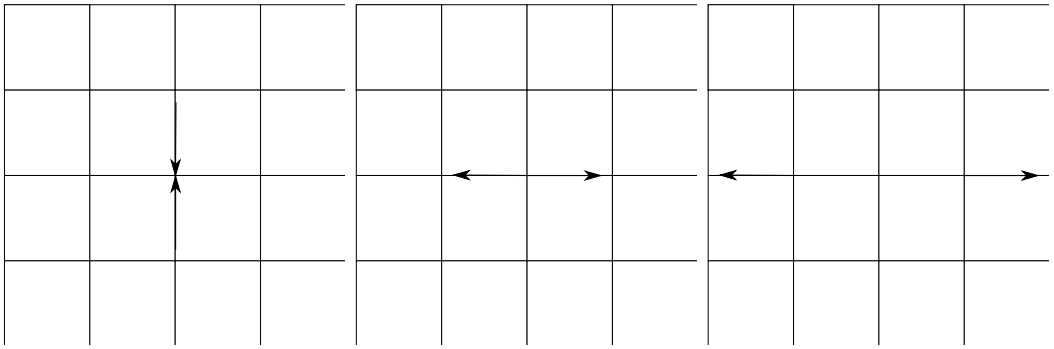


Figura 2.2: Dinâmica do HPP. (a)Estado de um nó antes da colisão. (b) Estado de um nó após a colisão. (c) Propagação das partículas de um nó.

As condições de contorno também devem ser tratadas no LGAC e elas podem ser de diferentes tipos. Uma condição de contorno, pode ser a condição de parede, que no autômato celular é chamada de reflexiva, onde uma partícula que chega em um nó de parede tem o sentido de sua velocidade invertida, como pode ser visto na Figura 2.3. Esta condição pode ser usada para as bordas do *lattice* ou até mesmo para representar obstáculos dentro do domínio.

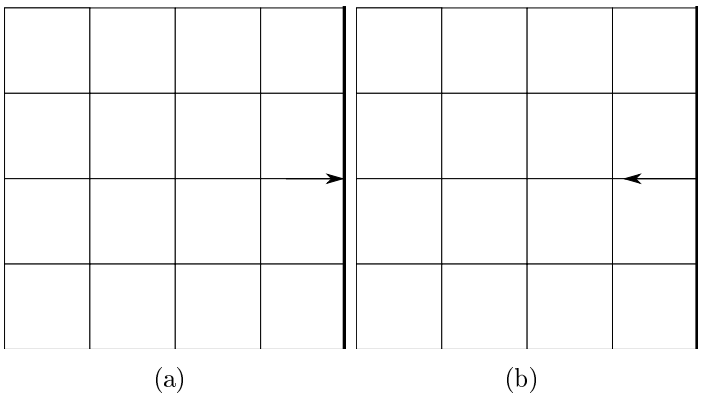


Figura 2.3: Condição de contorno reflexiva. (a) Partícula chegando em um nó do contorno no tempo t. (b) Partícula no tempo $t + \Delta t$, após a aplicação da condição de contorno.

Outra condição que pode ser usada é a condição de contorno periódica, onde a partícula que sai de um nó da borda entra em um nó da borda oposta, como pode ser visto na Figura 2.4.

O Algoritmo 1 descreve a dinâmica do método, onde é configurado qual modelo de LGAC será utilizado. Em seguida, os nós do *lattice* são inicializados de acordo com o problema a ser simulado e para um determinado tempo as etapas de colisão, contorno

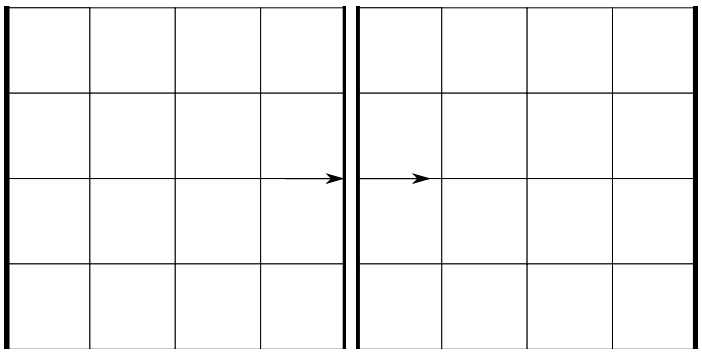


Figura 2.4: Condição de contorno Periódica. (a) Partícula chegando em um nó do contorno no tempo t. (b) Partícula em um nó do contorno no tempo $t + \Delta t$.

Algoritmo 1: Algoritmo do LGAC

```
1  Configuração dos parâmetros do modelo
2  Inicialização do LGAC
3  para k ← 1 até numIteracoes faça
4      t ← t + Δt
5      para cada nó n faça
6          Colisão: n_i = n_i + Δ_i
7      para cada nó do contorno faça
8          Condição de contorno
9      para cada nó n faça
10         Propagação: n_i(x + e_i, t + 1) = n_i(x, t)
11     Grava dados em arquivo
```

e propagação são realizadas. Primeiramente o operador de colisão é aplicado a todos os nós do *lattice* e as partículas podem sofrer mudanças na direção de deslocamento. Em seguida é feito o tratamento dos nós de contorno, de acordo com a condição de contorno imposta. Então é aplicado a propagação das partículas em todos os nós do *lattice*, o que desloca as partículas de um nó para nós da sua vizinhança.

Para se obter as variáveis macroscópicas do fluido, calcula-se a média dos estados de um conjunto de nós do *lattice*, que é chamada de f_i e pode assumir valores reais de 0 a 1. Os valores de f_i podem ser interpretados como a probabilidade de se encontrar uma partícula com velocidade e_i. Partindo de f_i pode-se chegar nas seguintes equações para a

densidade ρ e velocidade $\mathbf{u}$:

$$\rho(\mathbf{x}, t) = \sum_{i=0}^{l-1} f_i(\mathbf{x}, t) \tag{2.5}$$

$$\rho(\mathbf{x}, t)\mathbf{u}(\mathbf{x}, t) = \sum_{i=0}^{l-1} \mathbf{e}_i f_i(\mathbf{x}, t) \tag{2.6}$$

onde l é o número de direções de velocidade, para o caso do HPP $l = 4$.

Em geral, a solução encontrada possui ruídos, onde os valores em cada ponto do *lattice* tem uma alta variação no tempo. Apesar desta solução com ruídos apresentar informações físicas interessantes, busca-se um campo hidrodinâmico suave, como o campo de densidade ou velocidade do fluido.

É importante ressaltar que através de análise multi-escala (Wolf-Gladrow et al, 2000), é possível mostrar que um modelo LGAC é capaz de recuperar o comportamento macroscópico de equações que governam a dinâmica dos fluidos. Em particular, o modelo HPP não é capaz de representar o comportamento macroscópico das equações de Navier-Stokes, devido à simplicidade da estrutura do *lattice*.

Com o objetivo de recuperar tal comportamento surge o modelo FHP (Frisch et al, 1986). A principal mudança em relação ao HPP é a estrutura do *lattice*, que passa a ter seis direções de velocidades para cada nó. Este aumento no número de estados por nó, leva ao aumento das possibilidades de colisão e consequentemente uma maior complexidade no operador de colisão.

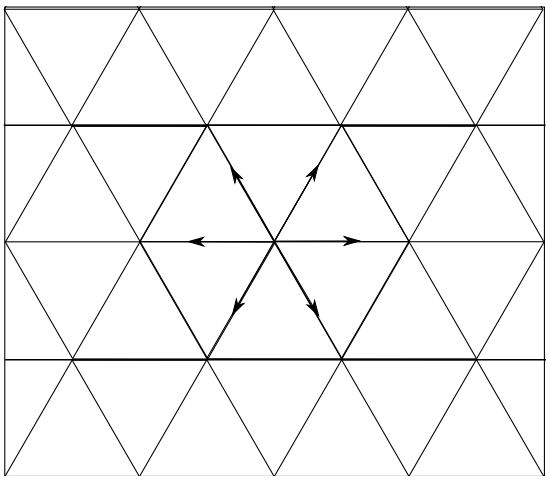


Figura 2.5: Modelo FHP

A Figura 2.5 mostra o modelo de *lattice* bidimensional para o FHP. Detalhes sobre

a implementação do modelo FHP e uma descrição detalhada da derivação das equações Navier-Stokes a partir do modelo FHP podem ser encontrados em (Wolf-Gladrow et al, 2000).

3 Método de Lattice Boltzmann para escoamentos de fluidos

Neste capítulo será apresentado o método de lattice Boltzmann, para escoamentos de fluidos. Primeiramente apresenta-se um modelo muito utilizado para dinâmica dos fluidos, as equações de Navier-Stokes, e em seguida são mostrados detalhes sobre o método de lattice Boltzmann, como a equação que descreve o método, condições de contorno e detalhes sobre a implementação do mesmo.

3.1 Equações de Navier-Stokes

Segundo (Franco et al, 2006, p. 429) um modelo matemático representa, muitas vezes de forma simplificada um fenômeno da natureza e sua solução simula propriedades dos processos naturais envolvidos, como a velocidade de escoamento de um fluido, a deformação de uma estrutura ou o crescimento de uma população. E as soluções das equações do modelo devem apresentar resultados compatíveis com o fenômeno físico modelado. Na maioria das vezes não é possível assumir hipóteses simplificadoras, de forma que o modelo tenha uma solução exata. Então parte-se para uma solução numérica do modelo através do uso do computador.

Muitas vezes os modelos matemáticos envolvem equações diferenciais, as quais tem o objetivo de descrever matematicamente fenômenos físicos que acontecem na natureza. Geralmente um modelo deve levar em conta vários aspectos do fenômeno, o que leva a um modelo com várias variáveis. Estes também podem possuir informações sobre a taxa de variação de suas variáveis, ou seja suas derivadas parciais. Chegando portanto à uma equação diferencial parcial ou a um sistema de equações diferenciais parciais. Uma equação diferencial parcial possui funções como incógnitas e envolve também as derivadas parciais destas funções.

As equações de Navier-Stokes são equações diferenciais parciais que constituem

um modelo muito utilizado para a simulação de problemas da dinâmica dos fluidos. Elas modelam o escoamento de fluidos compressíveis e incompressíveis, turbulentos e laminares (Fortuna et al, 2000, p. 227). As equações de Navier-Stokes representam princípios físicos tais como conservação de massa, conservação de momento - a taxa de variação do momento do fluido é igual a força resultante que atua sobre o fluido. E ainda, o princípio de conservação de energia, onde a taxa de variação da energia é igual a soma do fluxo de calor resultante para o fluido com o trabalho realizado sobre o fluido. Aplicando esses princípios de conservação pode-se chegar às equações de Navier-Stokes para o escoamento de um fluido incompressível, as quais são apresentadas a seguir.

A Equação 3.1 representa a conservação de massa, o único termo representa a taxa de variação de massa por unidade de volume. Esta equação também é chamada de equação da continuidade:

$$\nabla \cdot (\rho \mathbf{u}) = 0 \tag{3.1}$$

onde ρ é a densidade do fluido, $\mathbf{u}$ é a velocidade do fluido e $\nabla\cdot$ é o operador divergente.

A Equação 3.2 representa a conservação de momento, onde a força resultante que atua no fluido é igual a soma vetorial das forças gravitacionais, forças de pressão e forças viscosas:

$$\rho \frac{\partial \mathbf{u}}{\partial t} = \rho \mathbf{g} - \nabla p + \nu \nabla^2 \mathbf{u} \tag{3.2}$$

onde g é a aceleração da gravidade, ∇p é o gradiente da pressão, ν é a viscosidade do fluido e ∇^2 é o operador laplaciano.

Estas equações vem sendo resolvidas por métodos tradicionais como: o método dos elementos finitos (Hughes et al, 2000), método das diferenças finitas (Leveque et al, 2007) e o método dos volumes finitos (Leveque et al, 2002). Basicamente, nesses métodos as equações diferenciais parciais são discretizadas e transformadas em equações algébricas, que são resolvidas por métodos numéricos para a solução de sistemas de equações lineares.

Enquanto os métodos tradicionais partem de um modelo contínuo, o LGAC parte de um modelo discreto, onde o fluido é trocado por um conjunto de partículas, que podem se mover em várias direções e colidirem entre si. A dinâmica dessas partículas é capaz de recuperar o comportamento macroscópico das equações de Navier-Stokes. Já o método de

Lattice Boltzmann, trabalha entre a escala microscópica e a macroscópica, isto é, na escala mesoscópica. Ele assume que para cada nó do *lattice*, existem funções de distribuições que podem ser interpretadas como a probabilidade de se encontrar uma partícula com uma determinada velocidade. No MLB substituímos as variáveis booleanas de ocupação do LGAC por funções reais de distribuição de partículas, o que elimina o ruído encontrado no LGAC, pois assumimos que os valores das distribuições variam suavemente no espaço e no tempo.

3.2 Método de lattice Boltzmann

O método de lattice Boltzmann (MLB) é um método numérico baseado em equações cinéticas, que simulam a dinâmica do fluidos na escala mesoscópica afim de recuperar o seu comportamento macroscópico. Ele vem se tornando bastante popular para a simulação de escoamentos de fluidos, como descrito em (Mohamad et al, 2011; Golbert et al, 2009; Zou et al, 1997). Enquanto no LGAC os estados são discretos, no MLB os estados são distribuições de probabilidade para a velocidade de uma partícula.

3.2.1 Do LGAC para o MLB

O método de lattice Boltzmann se encontra na mesoescala, que considera funções de distribuição das partículas. Para migrarmos da microescala, onde são considerados valores discretos, para a mesoescala, onde são considerados valores reais, é necessário deixar de lado o movimento individual das micropartículas e considerar médias por amostra de regiões do *lattice*. Então, podemos definir $N_i = \langle n_i \rangle$ como médias calculadas por amostra da distribuição das partículas descrita pelas variáveis booleanas n_i. Aplicando este cálculo das médias à equação do LGAC, chega-se na seguinte equação:

$$N_i(\mathbf{x} + \mathbf{e}_i \Delta x, t + \Delta t) = N_i(\mathbf{x}, t) + \langle \Delta_i(\mathbf{n}) \rangle \tag{3.3}$$

onde Δx é o espaçamento e Δt é o passo de tempo.

O operador de colisão se tornou mais complexo, que passa a ser uma média tomada sobre produtos das variáveis n_i. Mas assumindo o princípio do caos molecular,

onde o movimento das partículas não é correlacionado antes da colisão, pode-se simplificar este termo. Assim, pode-se substituir a média do operador de colisão pelo termo de colisão aplicado à média da distribuição das partículas, chegando na equação abaixo:

$$N_i(\mathbf{x} + \mathbf{e}_i\Delta x, t + \Delta t) = N_i(\mathbf{x}, t) + \Delta_i(\mathbf{N}) \tag{3.4}$$

que é similar à equação do método de lattice Boltzmann, dada por:

$$f_i(\mathbf{x} + \mathbf{e}_i\Delta x, t + \Delta t) = f_i(\mathbf{x}, t) + \Omega_i(\mathbf{f}) \tag{3.5}$$

onde f_i são as funções de distribuição das partículas e Ω_i é o operador de colisão. A equação do MLB também pode ser derivada partindo-se da equação de Boltzmann, como será mostrado na próxima seção.

3.2.2 Da equação de Boltzmann para o MLB

O movimento de um fluido pode ser descrito em vários níveis. A forma mais básica de descrever o movimento de um fluido é a partir do movimento de suas partículas, que satisfazem os princípios de conservação de massa, momento e energia. Então é possível aplicar a segunda lei de Newton para calcular as variáveis macroscópicas envolvidas, como na equação abaixo:

$$\mathbf{F} = m\frac{d\mathbf{v}}{dt}, \quad \mathbf{v} = \frac{d\mathbf{x}}{dt} \tag{3.6}$$

onde $\mathbf{F}$ engloba forças externas e inter-moleculares, $\mathbf{v}$ é a velocidade da partícula, $\mathbf{x}$ a posição e t o tempo.

Entretanto, quando se considera sistemas grandes, fica impraticável calcular o movimento de todas as partículas envolvidas no sistema. Então pode-se considerar funções de distribuição de partículas, que são médias por amostra e indicam a probabilidade encontrar partículas em uma certa vizinhança com uma determinada velocidade em um determinado instante de tempo. É nesse contexto que surge a equação de Boltzmann, a qual descreve o comportamento estatístico de um sistema termodinâmico, como por exemplo, o movimento de um fluido com gradientes de temperatura no espaço e cujo

movimento vai de regiões mais quentes para regiões mais frias através do transporte de partículas.

A equação de transporte de Boltzmann é dada pela seguinte equação:

$$\frac{\partial f}{\partial t} + \mathbf{v} \cdot \nabla f + \frac{\mathbf{F}}{m} \frac{\partial f}{\partial \mathbf{v}} = Q(f) \tag{3.7}$$

e na ausência de forças de corpo, obtém-se a equação

$$\frac{\partial f}{\partial t} + \mathbf{v} \cdot \nabla f = Q(f) \tag{3.8}$$

onde $f(\mathbf{x}, \mathbf{v}, t)$ é a função de distribuição das partículas no espaço de fase contínuo, $\mathbf{x}$ é a posição, $\mathbf{v}$ é a velocidade das partículas e $Q(f)$ é o operador de colisão, que além de não-linear trata-se de um termo integral.

As variáveis macroscópicas do fluido, ρ e $\mathbf{u}$, são calculadas a partir dos momentos da distribuição f, e são dadas pelas equações:

$$\rho = \int f d\mathbf{v}, \quad \rho\mathbf{u} = \int \mathbf{v} f d\mathbf{v} \tag{3.9}$$

Um dos maiores problemas em lidar com a equação de Boltzmann é a estrutura complicada do operador de colisão. Sendo assim, uma alternativa muito utilizada é a aproximação BGK para este termo (Bhatnagar et al, 1954). Utilizando essa aproximação chega-se na seguinte equação:

$$\frac{\partial f}{\partial t} + \mathbf{v} \cdot \nabla f = \omega(f^{eq}(\mathbf{x}, \mathbf{v}, t) - f(\mathbf{x}, \mathbf{v}, t)) \tag{3.10}$$

onde ω é a frequência de colisão e f^{eq} é a função distribuição de equilíbrio, que será discutida adiante. Muitas vezes o coeficiente ω é escrito como $\omega = \frac{1}{\tau}$, onde τ é chamado de parâmetro de relaxamento. Para se chegar à equação do método, primeiramente o espaço de velocidades é discretizado em um conjunto discreto de velocidades $\mathbf{e}_i$, com $i = 0, 1, 2, 3, \ldots, l-1$, onde l é a dimensão do espaço do espaço de velocidade. Assim a equação de Boltzmann fica da seguinte forma:

$$\frac{\partial f_i}{\partial t} + \mathbf{e}_i \cdot \nabla f_i = \frac{1}{\tau}(f_i^{eq} - f_i), \quad i = 0, \ldots, l-1 \tag{3.11}$$

Podemos aproximar as derivadas no tempo e no espaço da equação anterior utilizando diferença finitas (He et al, 1997) e assim chega-se na equação que governa a dinâmica do método de Lattice-Boltzmann:

$$f_i(\mathbf{x} + \mathbf{e}_i\Delta t, t + \Delta t) - f_i(\mathbf{x}, t) = \frac{1}{\tau}(f_i^{eq}(\mathbf{x}, t) - f_i(\mathbf{x}, t)) \tag{3.12}$$

onde f_i é a função de distribuição das partículas na direção de velocidade $\mathbf{e}_i$, τ é o fator de relaxamento, Δx é o espaçamento da malha, Δt é o passo de tempo.

3.2.3 Aproximação BGK

A aproximação BGK se baseia no princípio de que cada colisão modifica a distribuição $f(\mathbf{x}, \mathbf{v}, t)$ em um valor proporcional à distância de f para uma distribuição de Maxwell, chamada de $f^{eq}(\mathbf{x}, \mathbf{v}, t)$. O novo operador de colisão pode ser expresso da seguinte forma:

$$Q'(f, f) = \frac{1}{\tau}(f^{eq}(\mathbf{x}, \mathbf{v}, t) - f(\mathbf{x}, \mathbf{v}, t)) \tag{3.13}$$

onde τ é fator de relaxamento do *lattice*. A distribuição de equilíbrio pode ser escrita da seguinte forma:

$$f^{eq} = \frac{\rho}{(\frac{2\pi}{3})^{\frac{D}{2}}} exp\left[-\frac{3}{2}(\mathbf{v} - \mathbf{u}) \cdot (\mathbf{v} - \mathbf{u})\right] \tag{3.14}$$

onde D é a dimensão do *lattice*, ρ é densidade do fluido e $\mathbf{u}$ é a velocidade macroscópica do fluido. Considerando que a velocidade do fluido é pequena em relação à velocidade do som, a função de distribuição de equilíbrio de Maxwell pode ser aproximada como:

$$f^{eq} = \frac{\rho}{(\frac{2\pi}{3})^{\frac{D}{2}}} exp\left[-\frac{3}{2}(\mathbf{v} \cdot \mathbf{v})\right]\left[1 + 3(\mathbf{v} \cdot \mathbf{u}) + \frac{9}{2}(\mathbf{v} \cdot \mathbf{u})^2 - \frac{3}{2}(\mathbf{u} \cdot \mathbf{u})\right] \tag{3.15}$$

Quando o espaço de velocidades é discretizado, a equação da distribuição de equilíbrio fica da seguinte forma:

$$f_i^{eq} = w_i\rho\left[1 + 3(\mathbf{e}_i \cdot \mathbf{u}) + \frac{9}{2}(\mathbf{e}_i \cdot \mathbf{u})^2 - \frac{3}{2}(\mathbf{u} \cdot \mathbf{u})\right] \tag{3.16}$$

onde w_i são coeficientes de peso que irão depender do modelo de lattice utilizado.

3.3 Modelos de Lattice

Vários modelos de *lattice*, ou malha, podem ser usados no MLB, sendo que em geral a terminologia usada é a DnQm, onde n é dimensão do *lattice* e m é a dimensão do espaço de velocidade.

Em uma dimensão do modelo mais popular é o D1Q3, que possui três funções de distribuição para cada nó do *lattice*. Ele considera a probabilidade de uma partícula se deslocar para a esquerda ou para a direita, além de considerar probabilidade da partícula não se mover. Os pesos w_i da distribuição de equilíbrio são:

$$w_0 = \frac{4}{6}, \;\; w_2 = \frac{1}{6}, \;\; w_3 = \frac{1}{6} \tag{3.17}$$

Em duas dimensões os modelos existem vários modelos como o D2Q5, que possui quatro direções de velocidade que as partículas podem assumir mais a probabilidade da partícula permanecer em repouso. Os pesos w_i possuem os seguintes valores:

$$w_0 = \frac{2}{6}, \;\; w_1 = \frac{1}{6}, \;\; w_2 = \frac{1}{6}, \;\; w_3 = \frac{1}{6}, \;\; w_4 = \frac{1}{6} \tag{3.18}$$

O modelo mais comum para duas dimensões é o D2Q9, o qual será utilizado em todos as simulações apresentadas neste trabalho, e que possui oito direções de velocidade mais a probabilidade de permanecer em repouso em um nó. Os pesos para este modelo são:

$$w_0 = \frac{4}{9}, \;\; w_{1-4} = \frac{1}{9}, \;\; w_{5-8} = \frac{1}{36} \tag{3.19}$$

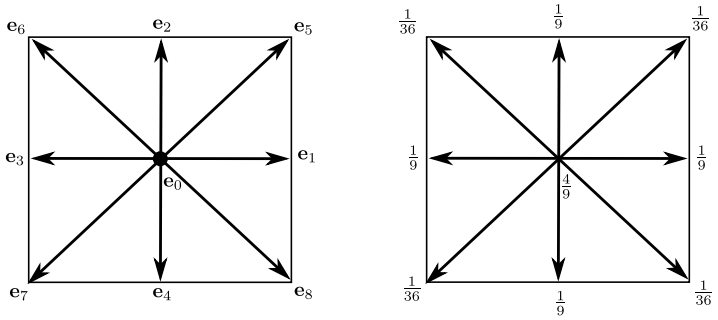


Figura 3.1: Modelo de lattice D2Q9 e seus coeficientes em cada direção.

A Figura 3.1 mostra a estrutura do modelo D2Q9, onde as variáveis macroscópicas do fluido, no MLB, podem ser recuperadas através das equações:

$$\rho = \sum_{i=0}^{l-1} f_i, \quad \rho\mathbf{u} = \sum_{i=0}^{l-1} e_i f_i \tag{3.20}$$

3.4 Análise multiescala, números de Mach e Reynolds

Este trabalho é voltado para a aplicação do método de lattice Boltzmann aos problemas da dinâmica dos fluidos e de reação-difusão, não tendo como foco a análise de erro entre o MLB e as equações de Navier-Stokes. Para tal seria necessário utilizar ferramentas matemáticas mais sofisticadas como a expansão multiescala de Chapman-Enskog na equação do MLB para derivar as equações de Navier-Stokes. Detalhes sobre a recuperação destas equações podem ser verifcados nos trabalhos (Golbert et al, 2009; Wolf-Gladrow et al, 2000; Dawnson et al, 1992; Blaak et al, 2000).

Seja $Ma = \frac{|\mathbf{u}|}{c_s}$, o número de Mach, que é a relação entre a velocidade do fluido e a velocidade do som no lattice. Sendo assim, a partir da Eq. (3.12) é possível recuperar as equações de Navier-Stokes incompressíveis, com ordens de aproximação de $\mathrm{O}\big(Ma^2\big)$ na equação da continuidade e $\mathrm{O}\big(M^3\big)$ na equação do momento, isto é:

$$\nabla \cdot \mathbf{u} = 0 + \mathrm{O}\big(Ma^2\big) \tag{3.21}$$

$$\rho\frac{\partial \mathbf{u}}{\partial t} = \rho\mathbf{g} - \nabla p + \nu\nabla^2 \cdot \mathbf{u} + \mathrm{O}\big(Ma^3\big). \tag{3.22}$$

Através da expansão multiescala de Chapman-Enskog, é possível mostrar ainda, a seguinte relação entre a viscosidade do fluido e o parâmetro de relaxação:

$$\nu = \frac{\Delta x^2}{3\Delta t}\left(\frac{1}{\tau} - \frac{1}{2}\right). \tag{3.23}$$

O número de Reynolds relaciona as forças de inércia e as forças viscosas, indicando se um escoamento é laminar ou turbulento e é dado pela seguinte equação:

$$Re = \frac{UL}{\nu}, \tag{3.24}$$

onde U é a velocidade macroscópica do fluido, L é o tamanho característico do escoamento e ν é a viscosidade do fluido. Dividindo a Equação 3.23 por UL, chega-se em

$$Ma = \frac{\Delta x}{L\sqrt{3}} \left(\frac{1}{\tau} - \frac{1}{2} \right) Re. \tag{3.25}$$

O valor de $\frac{L}{\Delta x}$ representa o número de nós, N, do *lattice* na direção do escoamento. Considerando Δx unitário chega-se em $L = N$. Então pode-se reescrever o número de Reynolds como $Re = \frac{UN}{\nu}$, que é chamado de número de Reynolds do *lattice*. Para se obter uma boa precisão com o MLB é necessário manter Ma pequeno, então deve-se escolher valores para U e τ que asseguram que Ma é pequeno. Em geral o valor de U deve estar entre 0.1 e 0.2, que não está relacionado com a velocidade macroscópica do fluido. Outra propriedade considerada é que os números de Reynolds macroscópico e Reynolds do *lattice* devem ser iguais. Então pode-se escolher valores arbitrários para U e ν dentro de uma faixa de valores, que asseguram a estabilidade do MLB.

3.5 Condições de contorno

A implementação das condições de contorno é parte essencial de qualquer método numérico. Dentro do contexto do MLB, existem diferentes formas de tratar as condições de contorno, como a condição de parede, chamada de reflexiva; condição periódica; condição de fluxo nulo, Neumann; e velocidade prescrita no contorno.

3.5.1 Reflexiva

A condição de contorno reflexiva (*bounce-back*) no MLB possui a mesma ideia do LGAC, onde a distribuição que chega em um nó do contorno tem seu sentido invertido. Como já dito esta condição pode ser usada para as bordas do *lattice* ou até mesmo para representar obstáculos dentro do domínio.

3.5.2 Periódica

A condição de contorno periódica também usa a mesma estratégia do LGAC, a distribuição que sai por um lado do contorno, entra no domínio novamente pela borda oposta a qual ela saiu.

3.5.3 Velocidade prescrita

Outra condição comum em problemas de escoamentos de fluidos é a prescrição de um determinado valor para a velocidade no contorno. Partindo das equações da densidade e do momento macroscópicos (Equação 3.20) é possível chegar à uma equação para esta condição de contorno. Para realizar os cálculos será considerado o modelo D2Q9, u será a velocidade horizontal e v a velocidade vertical. A densidade do fluido é dada por:

$$\rho = f_0 + f_1 + f_2 + f_3 + f_4 + f_5 + f_6 + f_7 + f_8 \tag{3.26}$$

Pela conservação de momento obtém-se:

$$\rho u = f_1 + f_5 + f_8 - f_3 - f_6 - f_7 \tag{3.27}$$

$$\rho v = f_5 + f_2 + f_6 - f_7 - f_4 - f_8 \tag{3.28}$$

Para exemplificar, vamos considerar um domínio retangular, como na Figura 3.2, onde as setas tracejadas indicam as distribuições desconhecidas no contorno. Vamos supor que desejamos prescrever a velocidade na borda esquerda do domínio, isto é, sendo u_w e v_w os valores da velocidade prescrita e ρ_w o valor da densidade que é desconhecida. Sendo assim, existem três equações e quatro incógnitas (ρ_w, f_1, f_5 e f_8). Assim chega-se nas seguintes equações:

$$\rho_w = f_0 + f_1 + f_2 + f_3 + f_4 + f_5 + f_6 + f_7 + f_8 \tag{3.29}$$

$$\rho_w u_w = f_1 + f_5 + f_8 - f_3 - f_6 - f_7 \tag{3.30}$$

$$\rho_w v_w = f_5 + f_2 + f_6 - f_7 - f_4 - f_8 \tag{3.31}$$

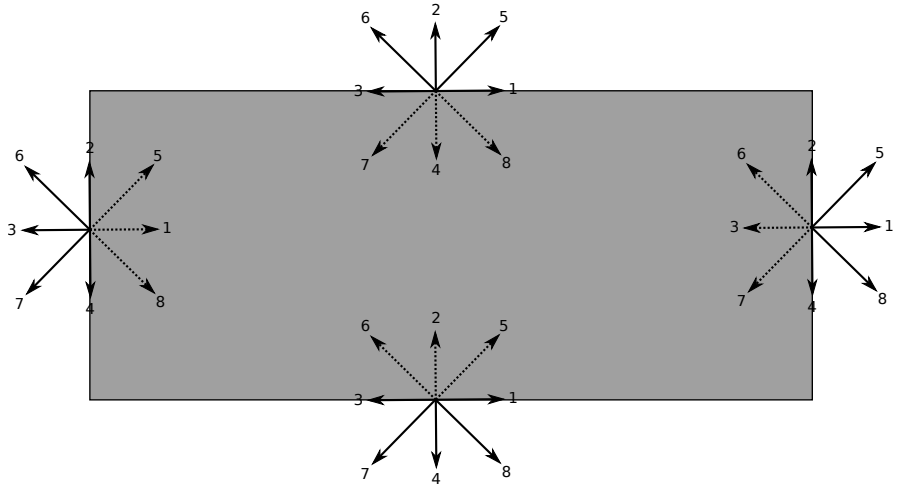


Figura 3.2: Distribuições conhecidas no contorno.

Para fechar o sistema precisamos de mais uma equação. (Zou et al, 1997) propuseram usar a seguinte equação:

$$f_1 - f_1^{eq} = f_3 - f_3^{eq} \tag{3.32}$$

a qual descreve a reflexão da parte de não-equilíbrio da função de distribuição normal à borda. As funções de equilíbrio podem ser calculadas por

$$f_1^{eq} = \frac{1}{9}\rho_w \left[1 + 3u_w + \frac{9}{2}u_w^2 - \frac{3}{2}(u_w^2 + v_w^2)\right] \tag{3.33}$$

$$f_3^{eq} = \frac{1}{9}\rho_w \left[1 - 3u_w + \frac{9}{2}u_w^2 - \frac{3}{2}(u_w^2 + v_w^2)\right] \tag{3.34}$$

substituindo essas equações na Equação 3.32, chega-se a

$$f_1 = f_3 + \frac{2}{3}\rho_w u_w \tag{3.35}$$

f_1 já foi resolvida pela equação anterior, substituindo f_1 nas outras equações, chega-se às

equações para as variáveis desconhecidas.

$$\rho_w = \frac{1}{1-u_w}(f_0 + f_2 + f_4 + 2(f_3 + f_6 + f_7)) \tag{3.36}$$

$$f_5 = f_7 - \frac{1}{2}(f_2 - f_4) + \frac{1}{6}\rho_w u_w + \frac{1}{2}\rho_w v_w \tag{3.37}$$

$$f_8 = f_6 + \frac{1}{2}(f_2 - f_4) + \frac{1}{6}\rho_w u_w - \frac{1}{2}\rho_w v_w \tag{3.38}$$

O mesmo cálculo pode ser feito para as outras bordas, o que mudará serão as variáveis desconhecidas e a condição de equilíbrio. Para mais detalhes, assim como para a decrição de como impor um valor de pressão no contorno, veja (Mohamad et al, 2011; Golbert et al, 2009).

3.6 Implementação

O Algoritmo 2 descreve o MLB para problemas de escoamento de fluidos. O MLB possui as mesmas etapas que o LGAC, mas no lugar de partículas com valores discretos, tem-se funções de distribuição de partículas. Na inicialização do método são dados os valores dos parâmetros do MLB, como: parâmetro de relaxamento do *lattice*, modelo de *lattice* com seus respectivos pesos e tamanho da malha. Além disso são passadas informações sobre o fluido, como densidade, viscosidade, número de Reynolds e as condições iniciais do problema, informando os valores da velocidade e densidade em cada ponto do *lattice*. Após a inicialização, realiza-se a colisão para todos os nós do lattice, onde é aplicado o operador de colisão BGK. Após esta etapa os nós possuem novos valores para suas distribuições, que devem ser propagadas para os nós vizinhos na etapa de propagação. Por exemplo, após a etapa de propagação a distribuição f_7 do nó (i, j) será igual à distribuição f_7 do nó $(i+1, j+1)$, e assim por diante. Em seguida é feito o tratamento das condições de contorno, de acordo com o tipo de condição imposta no problema.

O Algoritmo 3 mostra a dinâmica da etapa de colisão para o modelo D2Q9, onde primeiro são calculados a densidade e a velocidade macroscópicas em cada nó do *lattice*. Em seguida calcula-se as distribuições de equilíbrio em cada nó, utilizando a Equação 3.16 e os valores encontrados para a densidade e velocidade. Então o cálculo do operador

Algoritmo 2: Algoritmo do MLB para escoamentos de fluidos.

```
Configuração dos parâmetros do modelo
Inicialização do MLB
para k ← 1 até numIteracoes faça
    t ← t + Δt
    para cada nó x faça
        Colisão: f_i^*(x,t) = f_i(x,t) + Ω(x,t)
    para cada nó x faça
        Propagação: f_i(x + e_i Δt, t + Δt) = f_i^*(x,t)
    para cada nó do contorno faça
        Condição de contorno
    Grava dados em arquivo
```

de colisão BGK é feito e armazenado em uma variável auxiliar f^*, que será utilizada na etapa de propagação.

Algoritmo 3: Algoritmo da etapa de colisão do MLB

```
para i ← 0 até nx faça
    para j ← 0 até ny faça
        ρ = f0[i,j] + f1[i,j] + f2[i,j] + f3[i,j] + f4[i,j] + f5[i,j] +
        f6[i,j] + f7[i,j] + f8[i,j];
        u^x = (f1[i,j] + f3[i,j] + f5[i,j] − f6[i,j] − f7[i,j] + f8[i,j])/ρ;
        u^y = (f2[i,j] − f4[i,j] + f5[i,j] + f6[i,j] − f7[i,j] − f8[i,j])/ρ;
        Calcula f_eq[i,j] usando a Eq. 3.16
        f0_tmp[i,j] = (1/τ) f0_eq[i,j] + (1 − 1/τ) f0[i,j]
        f1_tmp[i,j] = (1/τ) f1_eq[i,j] + (1 − 1/τ) f1[i,j]
        f2_tmp[i,j] = (1/τ) f2_eq[i,j] + (1 − 1/τ) f2[i,j]
        f3_tmp[i,j] = (1/τ) f3_eq[i,j] + (1 − 1/τ) f3[i,j]
        f4_tmp[i,j] = (1/τ) f4_eq[i,j] + (1 − 1/τ) f4[i,j]
        f5_tmp[i,j] = (1/τ) f5_eq[i,j] + (1 − 1/τ) f5[i,j]
        f6_tmp[i,j] = (1/τ) f6_eq[i,j] + (1 − 1/τ) f6[i,j]
        f7_tmp[i,j] = (1/τ) f7_eq[i,j] + (1 − 1/τ) f7[i,j]
        f8_tmp[i,j] = (1/τ) f8_eq[i,j] + (1 − 1/τ) f8[i,j]
```

O Algoritmo 4 apresenta como é feita a etapa de propagação no modelo D2Q9, onde as distribuições de cada nó do *lattice*, que acabaram de sofrer colisão (f_{tmp}), propagam para um dos nós vizinhos. No contorno algumas distribuições podem ter índices inválidos, que devem ser tratados.

Algoritmo 4: Algoritmo da etapa de propagação do MLB

```
1  para i ← 0 até nx faça
2      para j ← 0 até ny faça
3          f0[i, j] = f0_tmp[i, j]
4          f1[i, j] = f1_tmp[i − 1, j]
5          f2[i, j] = f2_tmp[i, j − 1]
6          f3[i, j] = f3_tmp[i + 1, j]
7          f4[i, j] = f4_tmp[i, j + 1]
8          f5[i, j] = f5_tmp[i − 1, j − 1]
9          f6[i, j] = f6_tmp[i + 1, j − 1]
10         f7[i, j] = f7_tmp[i + 1, j + 1]
11         f8[i, j] = f8_tmp[i − 1, j + 1]
```

4 Método de Lattice Boltzmann para problemas de reação-difusão

Equações do tipo reação-difusão são equações diferenciais parciais que envolvem a difusão de uma determinada variável acoplada a um sistema de variáveis secundárias que influenciam o comportamento da variável de interesse, como o exemplo na seguinte equação:

$$\frac{\partial f}{\partial t} = \nabla \cdot (\Gamma \nabla f) + R(f) \tag{4.1}$$

onde f é a variável de interesse, Γ é o coeficiente de difusão e $R(f)$ representa a parte reativa da equação.

Este tipo de equação tem sido utilizado para modelar diferentes problemas, como o crescimento de tumores, a dispersão de fogo em uma floresta incendiada e na modelagem da eletrofisiologia cardíaca. Este capítulo apresentará como o método de Lattice Boltzmann pode ser usado para resolver problemas do tipo reação-difusão, em particular, aplicando-o na resolução de um modelo da eletrofisiologia cardíaca.

4.1 Modelagem da eletrofisiologia cardíaca

Recentes avanços nos campos da biologia computacional e de imagens médicas possibilitaram o desenvolvimento de modelos computacionais multiescala personalizados para dados obtidos de pacientes, através dos quais importantes contribuições na compreensão de diversos fenômenos complexos da atividade elétrica cardíaca, como por exemplo a formação de ondas de reentrada associadas com a desordem do ritmo cardíaco, foram obtidas recentemente (Beaumont et al, 1998; Nash et al, 2004).

Em geral esses fenômenos são altamente complexos e possuem uma natureza multiescala, já que a escala temporal abrange fenômenos que variam desde o microssegundo ao minuto, enquanto a escala espacial varia desde o nível celular (micrometro) ao nível do órgão (centímetro). Dentro desse contexto, os modelos computacionais (Plonsey, 1988)

se tornaram ferramentas extremamente valiosas para o estudo desses fenômenos, pois são capazes de agregar informações de experimentos em diferentes escalas para se obter um melhor entendimento global da atividade elétrica cardíaca.

Naturalmente a complexidade dos processos biofísicos se traduz em modelos matemáticos e computacionais complexos, os quais são descritos por sistemas de equações diferenciais parciais (EDPs) acoplados a sistemas de equações diferenciais ordinárias (EDOs), resultando em problemas com milhões de variáveis e muitos parâmetros. O modelo mais utilizado para descrever a atividade elétrica do tecido cardíaco é o modelo do bidomínio (Plonsey, 1988). Muitas vezes, um modelo mais simples cuja solução numérica é menos custosa, denominado modelo do monodomínio (Sundnes, 2006), é utilizado para realizar as simulações. Entretanto em muitas situações é preciso escolher entre o nível de detalhes usados nos modelos e os recursos computacionais disponíveis, para que as simulações sejam tratáveis e executadas em um tempo razoável. Por exemplo, muitas vezes a simulação computacional de geometrias complexas como a dos ventrículos esquerdo e direito, representam um grande desafio computacional, e nesse caso pode ser preciso escolher simplificar o nível de detalhes presentes na discretização da geometria ou até mesmo na precisão da solução numérica obtida.

4.1.1 Modelo do monodomínio

No tecido cardíaco, a onda de excitação se espalha através do mesmo pois as células cardíacas são eletricamente acopladas através de proteínas especiais chamadas junções *gap*. Este fenômeno pode ser descrito matematicamente por uma equação de reação-difusão denominada modelo monodomínio, a qual é dada por:

$$\beta C_m \frac{\partial v}{\partial t} = \nabla \cdot (\boldsymbol{\sigma} \nabla v) - \beta I_{ion}(v, \boldsymbol{\eta}), \tag{4.2}$$

onde v é a variável de interesse e representa o potencial transmembrânico, β é a razão superfície-volume das células cardíacas, C_m é a capacitância da membrana, I_{ion} é a densidade total da corrente iônica que é função de v e de um vetor de variáveis de estado $\boldsymbol{\eta}$, I_{stim} é uma corrente estímulo aplicada e $\boldsymbol{\sigma}$ é o tensor de condutividade. Por simplicidade,

no presente trabalho, iremos considerar que $\boldsymbol{\sigma} = \sigma\mathbf{I}$, isto é, o tensor de condutividade é isotrópico.

Para completar o modelo descrito pela Eq. (4.2), precisamos ainda especificar condições iniciais e condições de contorno apropriadas. As condições iniciais para $v = v(\mathbf{x}, t)$ e $\boldsymbol{\eta} = \boldsymbol{\eta}(\mathbf{x}, t)$ são dadas por:

$$v(\mathbf{x}, 0) = v_0(\mathbf{x}) \tag{4.3}$$

$$\boldsymbol{\eta}(\mathbf{x}, 0) = \boldsymbol{\eta}_0(\mathbf{x}). \tag{4.4}$$

Nesse trabalho, assumimos que o tecido é isolado em suas fronteiras, isto é, condições de contorno de fluxo nulo são impostas sobre v ao longo de todas as superfícies do miocárdio:

$$\mathbf{n} \cdot \boldsymbol{\sigma} \nabla v = 0, \tag{4.5}$$

onde $\mathbf{n}$ é um vetor unitário normal à superfície do tecido.

4.1.2 Modelos celulares

Para completar a descrição da Eq. (4.2) é preciso ainda especificar o termo de reação $I_{ion}(v, \boldsymbol{\eta})$, o qual descreve a corrente iônica total por unidade de área da membrana celular, isto é, a soma de diversas correntes através de diferentes canais iônicos, como por exemplo, o canal de potássio (K^+), de sódio (Na^+), e de cálcio (Ca^{2+}).

Existem diversos modelos celulares (ou modelos iônicos) disponíveis na literatura que descrevem o comportamento de células cardíacas do átrio, ventrículo, fibras de Purkinje, etc, em diferentes espécies. Nesse contexto, o modelo mais conhecido é o de Hodgkin-Huxley, que descreve o potencial de ação em células do axônio de lula, onde apenas três correntes iônicas são consideradas: a corrente de sódio, de potássio e uma terceira corrente de fuga. Mais detalhes podem ser encontrados em Hodgkin et al (1952).

Nesse trabalho iremos apresentar simulações da atividade elétrica cardíaca utilizando dois modelos celulares diferentes. O primeiro modelo, de Mitchell-Schaeffer (MS), é um modelo simplificado, que através de apenas duas variáveis busca reproduzir o comportamento das células cardíacas de forma qualitativa (Mitchell et al, 2003). Por outro

lado, o modelo de Luo-Rudy (LR) trata-se de um modelo mais detalhado que representa de forma quantitativa, incluindo a atividade de diferentes correntes iônicas, a geração do potencial de ação de células ventriculares do preá-da-índia (Luo et al, 1991).

4.1.3 Modelo de Mitchell-Schaeffer

O modelo de Mitchell-Schaeffer é dado pelo seguinte conjunto de EDOs:

$$\frac{dv}{dt} = I_{in}(v,h) + I_{out}(v) + I_{stim}(t), \tag{4.6}$$

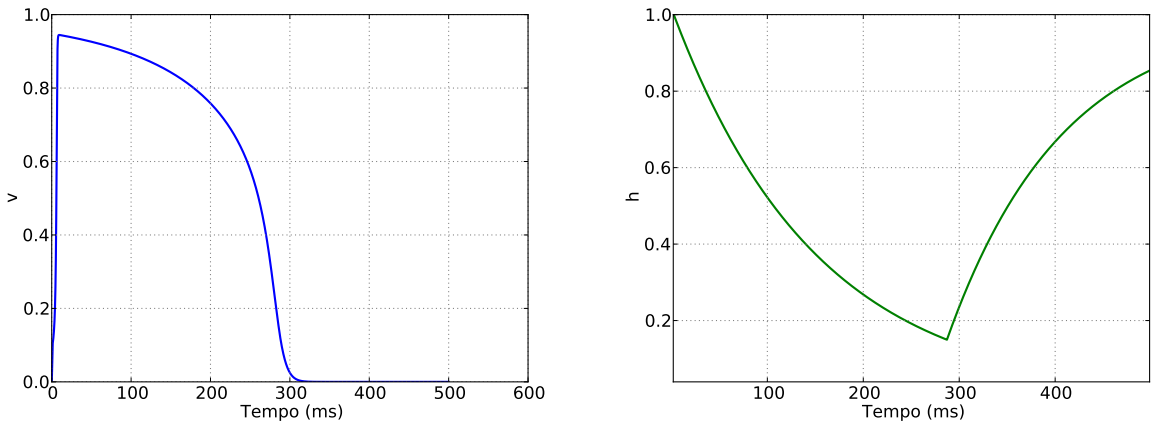
$$\frac{dh}{dt} = \begin{cases} \frac{1-h}{\tau_{open}} & \text{se } v < v_{gate} \\ \frac{-h}{\tau_{close}} & \text{se } v > v_{gate} \end{cases} \tag{4.7}$$

onde τ_{in}, τ_{out}, τ_{open}, τ_{close} e v_{gate} são constantes, e as correntes I_{in} e I_{out} são dadas por

$$I_{in}(v,h) = \frac{hC(v)}{\tau_{in}} \tag{4.8}$$

$$I_{out}(v) = -\frac{v}{\tau_{out}} \tag{4.9}$$

onde $C(v)$ é uma função cúbica dada por $C(v) = v^2(1-v)$ e I_{stim} é uma corrente de estímulo aplicada. A Figura 4.1 apresenta o gráfico das variáveis v e h ao longo do tempo. Veja mais detalhes sobre o modelo em Mitchell et al (2003).



(a) (b)

Figura 4.1: Exemplo de simulação do modelo de Mitchell-Schaeffer usando o método de Euler explícito durante 500 ms, inicializado com um estímulo inicial aplicado de $I_{stim} = 0.1$. (a) Potencial transmembrânico (variável v) e (b) variável h.

4.1.4 Modelo de Luo-Rudy

Além do modelo de Mitchell-Schaeffer de duas variáveis, que descreve o comportamento da dinâmica da membrana de células cardíacas de forma qualitativa, usamos também o modelo Luo-Rudy, o qual descreve a atividade elétrica em células ventriculares.

No modelo LR-I, o termo I_{ion} é definido como:

$$I_{ion}(v, \boldsymbol{\eta}) = I_{Na} + I_{si} + I_K + I_{K1} + I_{Kp} + I_b, \tag{4.10}$$

onde I_{Na} é a corrente rápida de sódio, I_{si} é a corrente lenta de entrada, I_K é a corrente de potássio dependente do tempo, I_{K1} é a corrente de potássio independente do tempo, I_{Kp} é a corrente de potássio plateau e I_b é uma corrente de fundo independente do tempo.

Quatro das seis correntes na Eq. (4.10) são controladas por variáveis descritas por EDOs que são da forma:

$$\frac{dn}{dt} = \frac{n_\infty - n}{\tau_n} \tag{4.11}$$

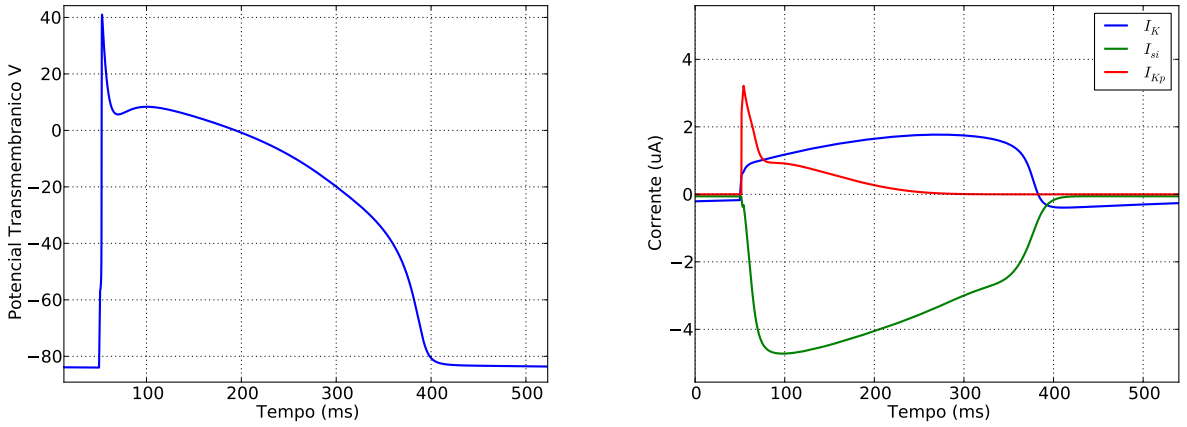
onde os termos n_∞ e τ_n são definidos como

$$n_\infty = \frac{\alpha}{\alpha + \beta}, \quad \tau_n = \frac{1}{\alpha + \beta}, \tag{4.12}$$

sendo que α e β são funções de v. A Figura 4.2(a) apresenta o potencial de ação, enquanto a Figura 4.2(b) apresenta algumas das correntes iônicas descritas na Eq. (4.10) do modelo LR. A descrição completa do sistema de EDOs, suas variáveis e parâmetros pode ser encontrada em Luo et al (1991).

4.2 Método de lattice Boltzmann para o modelo monodomínio

Para simular a dinâmica do modelo de reação-difusão descrito pela Eq. (4.2), iremos utilizar o método de lattice Boltzmann. A função de distribuição da espécie v no tempo t, posição $\mathbf{x}$ com velocidade $\mathbf{e}_i$ é denotada por $f_i(\mathbf{x}, t)$. A equação de lattice Boltzmann



(a) (b)

Figura 4.2: Exemplo de simulação do modelo de Luo-Rudy usando o método de Euler explícito durante 500 ms, inicializado com um estímulo inicial aplicado em $t = 5$ ms. (a) Potencial transmembrânico v e (b) algumas correntes iônicas (I_K, I_{si} e I_{Kp}) do modelo de Luo-Rudy.

para $f_i(\mathbf{x}, t)$ pode ser escrita como:

$$f_i(\mathbf{x} + \mathbf{e}_i, t + \Delta t) - f_i(\mathbf{x}, t) = \Omega(\mathbf{x}, t), \tag{4.13}$$

onde $\Omega(\mathbf{x}, t)$ é o operador de colisão para v, o qual depende das funções de distribuição $f_i(\mathbf{x}, t)$. Como descrito em (Dawnson et al, 1992), para problemas de reação-difusão, este termo pode ser escrito como a soma de um termo reativo Ω^R e de um termo não-reativo denotado por Ω^{NR}.

A parte não-reativa da colisão é descrita usando a tradicional aproximação BGK (Bhatnagar-Gross-Krook) (Bhatnagar et al, 1954), a qual é dada por:

$$\Omega^{NR}(\mathbf{x}, t) = -\frac{1}{\tau}(f_i^{eq}(\mathbf{x}, t) - f_i(\mathbf{x}, t)), \tag{4.14}$$

onde τ é o parâmetro de relaxação e f_i^{eq} a função de distribuição de equilíbrio, a qual depende da variável v e da velocidade $\mathbf{u}$. Em geral, quando o método de lattice Boltzmann é aplicado para a simulação de fluidos através da equação de Navier-Stokes, a função de distribuição de equilíbrio f_i^{eq} é dada pela Eq. 3.16. Entretanto, como estamos interessados em modelar um problema de reação-difusão pura, vamos considerar que temos velocidade

nula, isto é, $\mathbf{u} = 0$. Nesse caso, a distribuição de equilíbrio se reduz à seguinte equação:

$$f_i^{eq}(\mathbf{x}, t) = w_i \, v. \tag{4.15}$$

onde v é o potencial de ação dado pela equação 4.19. Para a parte reativa do operador de colisão, seguindo (Dawnson et al, 1992) vamos assumir que R representa a mudança na densidade da espécie v devido à reação, então a parte reativa será distribuída entre as diferentes direções de forma proporcional aos pesos w_i, isto é:

$$\Omega^R(\mathbf{x}, t) = w_i R. \tag{4.16}$$

A forma exata do termo R precisa ser especificada, e como veremos, para o problema da propagação elétrica no tecido cardíaco, R irá depender do modelo celular utilizado, sendo que nesse trabalho usaremos o modelo de Mitchell-Schaeffer e o modelo de Luo-Rudy.

4.2.1 Condição de contorno

Para impor as condições de contorno do tipo fluxo nulo (condição de contorno do tipo Neumann), vamos usar a mesma abordagem adotada em (Mohamad et al, 2011), onde a condição de fluxo nulo é imposta através de uma aproximação por diferenças finitas. Sem perda de generalidade, vamos assumir que $\mathbf{n} = (-1, 0)$, e nesse caso a condição de contorno é dada por:

$$\frac{\partial v}{\partial x} = 0, \tag{4.17}$$

vamos considerar ainda que a parte do domínio onde queremos impor fluxo nulo esteja situada em $\mathbf{x} = (x_0, y_0)$ e que, para $x > x_0$ o valor de v seja conhecido. Aproximando $\frac{\partial v}{\partial x}$ por diferenças finitas, é possível mostrar que as seguintes equações:

$$f_i(x_0, y_0) = f_i(x_0 + \Delta x, y_0), \quad \text{para} \quad i = 1, \ldots, 8, \tag{4.18}$$

aproximam a condição de contorno de fluxo nulo com precisão de primeira ordem (Mohamad et al, 2011). O mesmo desenvolvimento pode ser feito para prescrever a condição de fluxo nulo em outras partes do contorno.

Para relacionar os resultados obtidos resolvendo a Eq. (4.13) com a solução da Eq. (4.2), usamos os momentos das funções de distribuição f_i. O momento de ordem zero relaciona as funções de distribuição f_i's com o potencial transmembrânico v, através da seguinte relação:

$$v(\mathbf{x}, t) = \sum_i f_i(\mathbf{x}, t). \tag{4.19}$$

O termo de reação R que aparece no operador de colisão da parte reativa Ω^R é determinado pelo modelo celular utilizado para descrever a dinâmica da célula. De forma geral os modelos celulares podem ser descritos como um sistema não-linear de EDOs:

$$C_m \frac{dv}{dt} = -I_{ion}(v, \boldsymbol{\eta}) + I_{stim} \tag{4.20}$$

$$\frac{d\boldsymbol{\eta}}{dt} = g(v, \boldsymbol{\eta}) \tag{4.21}$$

onde $g(v, \boldsymbol{\eta})$ depende de v e de um vetor de variáveis de estado $\boldsymbol{\eta}$ que são descritas por EDOs. Um exemplo completo é dado pelas Eqs. (4.6) e (4.7) do modelo MS.

Sendo assim, como a Eq. (4.13) do método de lattice Boltzmann governa a dinâmica do potencial transmembrânico v, temos que o termo reativo R, acoplado a esta equação, é dado pelo termo I_{ion} do modelo celular utilizado. Em particular, para o modelo MS o termo reativo R, será denotado por R^{MS}, o qual é dado por:

$$R^{MS} = I_{ion} = I_{in} + I_{out} + I_{stim}, \tag{4.22}$$

enquanto que para o modelo de Luo-Rudy, tendo em vista a Eq. (4.10), o termo reativo, denotado por R^{LR} é dado por:

$$R^{LR} = I_{ion} = I_{Na} + I_{si} + I_K + I_{K1} + I_{Kp} + I_b. \tag{4.23}$$

Nesse trabalho, utilizamos o método de lattice Boltzmann para resolver a equação do monodomínio em problemas bidimensionais (2D). Sendo assim, utilizamos um grid cartesiano com um lattice do tipo D2Q9, o qual possui 8 direções de movimento mais a possibilidade de manter a distribuição de partículas parada. Para este caso, pode-se mostrar (Mohamad et al, 2011; Golbert et al, 2009) que os coeficientes (pesos) $w_i, i = 0, \ldots, 8$ associados às funções de distribuição em cada direção $\mathbf{e}_i$ (veja Figura 3.1) são dados por

$$w_0 = \frac{4}{9}, \quad w_{1-4} = \frac{1}{9}, \quad w_{5-8} = \frac{1}{36}, \tag{4.24}$$

e ainda, utilizando uma expansão multiescala de Chapman-Enskog, é possível mostrar a seguinte relação entre o coeficiente de difusão (condutividade) e o parâmetro de relaxação:

$$\sigma = \frac{\Delta x^2}{3\Delta t}\left(\frac{1}{\tau} - \frac{1}{2}\right). \tag{4.25}$$

Através da expansão multiescala de Chapman-Enskog, é possível mostrar que o método de lattice Boltzmann recupera as equações macroscópicas de reação-difusão. Entretanto, essa etapa não será apresentada nesse trabalho, e pode ser verificada em outros trabalhos (Dawnson et al, 1992; Blaak et al, 2000).

4.2.2 Implementação computacional

Para realizar a solução numérica da Eq. (4.2) através do método de lattice Boltzmann, associamos a cada nó do *lattice* um modelo celular que descreve a dinâmica da célula cardíaca. Por exemplo, para uma simulação usando o modelo MS, cada nó, além da variável v que é a variável de interesse cuja evolução é descrita pelo MLB, temos também a variável de estado h do modelo celular MS.

A solução numérica do MLB aplicado ao problema de reação-difusão da eletrofisiologia cardíaca é apresentada no Algoritmo 5. Inicialmente, os parâmetros do método e do modelo são inicializados, em seguida os vetores do MLB e das EDOs são alocados e inicializados com seus valores iniciais, isto é, todos os nós do *lattice* são inicializados com os valores da condição inicial do modelo celular adotado. O próximo passo é o *loop*

no tempo, onde a cada passo as etapas de colisão, propagação e condição de contorno do MLB são aplicadas. Além disso, após a atualização da variável v que é feita na etapa da colisão do MLB, as variáveis de estado $\boldsymbol{\eta}$ são atualizadas utilizando o método de Euler explícito.

Algoritmo 5: Algoritmo do MLB aplicado ao problema do monodomínio

```
1  Configuração dos parâmetros do modelo
2  Inicialização do MLB
3  Inicialização das EDOs
4  para k ← 1 até numIteracoes faça
5      t ← t + Δt
6      para cada nó x faça
7          Colisão: f_i^*(x, t) = f_i(x, t) + Ω(x, t)
8          Atualiza variáveis de estado η(x, t) usando Euler explícito
9      para cada nó x faça
10         Propagação: f_i(x + e_i, t + Δt) = f_i^*(x, t)
11     para cada nó do contorno faça
12         Condição de contorno usando Eq. (4.18)
13     Grava dados em arquivo
```

5 Técnicas de computação paralela

A simulação de problemas de interesse científico tem demandado cada vez mais poder computacional, para que seja resolvida o mais rápido possível. Entretanto o desempenho dos processadores atuais não tem aumentado tanto de uma geração para outra, como no passado, pois aumentar o desempenho de um processador é uma tarefa cada vez mais difícil. Uma alternativa a este problema é o uso de computação paralela, que busca dividir a computação de um problema a ser tratado entre vários processadores, com o objetivo de diminuir o tempo de execução. Simular a atividade elétrica do coração em tempo real seria muito interessante no tratamento de um paciente com doenças cardíacas e a programação paralela pode contribuir para diminuir o tempo gasto para a simulação deste e de outros problemas. Neste capítulo será apresentado de forma sucinta as principais arquiteturas de computação paralela, assim como as ferramentas utilizadas para desenvolver as versões paralelas das implementações deste trabalho, MPI e CUDA.

5.1 Arquiteturas de Computação Paralela

Os dois principais paradigmas de programação paralela são os sistemas de memória compartilhada e os sistemas de memória distribuídas, que necessitam de estratégias diferentes para a implementação.

5.1.1 Memória Compartilhada

Em um sistema de memória compartilhada, como o próprio nome diz, os processadores compartilham a memória e eles podem ler e escrever na memória (Pacheco, 2011), como pode ser visto na Figura 5.1. Devido a este compartilhamento é necessária uma supervisão para que dois processadores diferentes escrevam no mesmo endereço de memória. A plataforma CUDA (NVIDIA, 2013), que descrevemos a seguir é uma plataforma que utiliza a arquitetura de memória compartilhada.

Este sistema é controlado por um único sistema operacional e os processado-

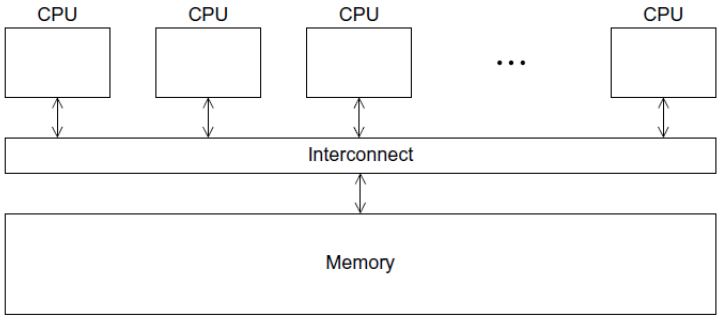


Figura 5.1: Sistema de memória compartilhada. Extraída de Pacheco (2011).

res que possuem vários núcleos, tem seus núcleos tratados como processadores distintos, onde todos estes processadores tem acesso a um mesmo espaço de endereçamento. Com esta arquitetura a comunicação entre processos é mais rápida, pois a memória está mais perto do processador fisicamente e a forma de programação paralela fica mais próxima à programação usual. Mas podem ocorrer problemas quando se tem concorrência no barramento, levando a uma latência devido a limitação de banda de memória. Outro problema pode ser o falso compartilhamento, que acontece quando dois ou mais processadores estão acessando endereços diferentes, mas que estão no mesmo bloco de cache. Quando um processador escreve no endereço, devido a coerência de cache, o dado será invalidado para os demais processadores que estão usando aquele dado, o que gera atraso no acesso ao dado daquela posição de memória. Além disso, o número de tarefas é limitado ao número de processadores da máquina.

5.1.2 Memória Distribuída

Em um sistema de memória distribuída, cada processador possui sua própria memória particular e deve se comunicar por mensagens através da rede, para que outros processadores possam ter acesso à sua memória Pacheco (2011), conforme a Figura 5.2. Cada processador opera sobre sua memória e as mudanças feitas por este processador só afetam a sua memória. O padrão MPI se baseia na arquitetura de memória distribuída (Quinn, 2003).

Nesta arquitetura existem processadores em máquinas diferentes se comunicando por passagem de mensagens pela rede. Então a rede de comunicação se torna uma parte importante do sistema, pois quanto mais lenta a rede mais tempo será gasto com comu-

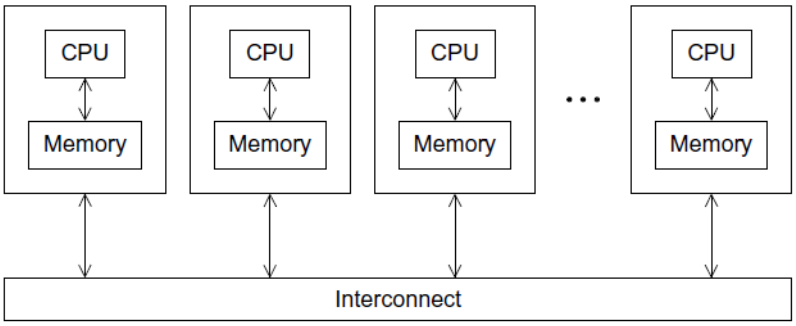


Figura 5.2: Sistema de memória distribuída. Extraída de Pacheco (2011)

nicação e o tempo gasto com comunicação afeta negativamente o desempenho da implementação paralela. Mas geralmente estes sistemas possuem uma rede de baixa latência, que utiliza protocolos com o objetivo de minimizar o tempo gasto com comunicação. Eles possuem alta escalabilidade, pois adicionando novas máquinas ao sistema o seu poder computacional aumenta e mais tarefas podem ser realizadas em paralelo, mas a implementação fica mais complexa.

5.2 Métricas de desempenho

Uma forma de medir o desempenho da aplicação paralela desenvolvida é através do cálculo de seu *speedup* ou aceleração, que é razão entre o tempo de execução sequencial e o tempo de execução paralelo, ou seja,

$$S = \frac{T_{sequencial}}{T_{paralelo}} \tag{5.1}$$

O desempenho ideal de uma aplicação paralela seria o linear, isto é, o *speedup* é igual ao número de processos. Mas na prática é muito difícil de se conseguir tal desempenho, pois muito tempo pode ser gasto com comunicação entre processos, ou *threads* precisam acessar uma região crítica, entre outros problemas. O *speedup* também pode ser super-linear, que pode acontecer quando o problema tem pouca comunicação e ainda aproveita do bom uso da cache, devido ao tipo de armazenamento na memória.

Outra forma de avaliar o desempenho de uma aplicação paralela é através da

eficiência, que é dada pela seguinte equação:

$$E = \frac{S}{N} \tag{5.2}$$

onde S é o *speedup* alcançado e N é o número de processos/*threads* utilizados.

5.2.1 Lei de Amdahl

Nem sempre é possível paralelizar totalmente um código e a partir desta observação surgiu a Lei de Amdahl que diz qual é a aceleração máxima alcançada quando o código pode ser parcialmente paralelizado. Por mais perfeita que seja uma implementação paralela, se o código possui uma parte que não pode ser paralelizada, dificilmente consegue-se chegar a um *speedup* linear. Segundo a Lei de Amdahl o *speedup* máximo que se pode conseguir com uma implementação paralela é dado pela seguinte equação:

$$S \leq \frac{1}{(1-P) + \frac{P}{N}} \tag{5.3}$$

onde P é a porcentagem do código que pode ser paralelizada e N é o número de processos/*threads* utilizados.

5.3 MPI

Como já apresentado anteriormente em um sistema de memória distribuída existe uma coleção de pares de processador-memória, onde cada processador só tem acesso à sua memória e para ter acesso a dados de outros processadores é necessário a comunicação por passagem de mensagens. Uma aplicação de memória distribuída é constituída de processos executando em máquinas distintas, sendo que os processos utilizam funções de envio e recebimento para se comunicar através da rede. O MPI, *Message Passing Interface*, é um padrão para este tipo de comunicação e baseado neste padrão bibliotecas foram desenvolvidas para as linguagens de programação, como a biblioteca MPICH para a linguagem C/C++ (Groop et al, 1994).

O MPI possui funções de comunicação ponto a ponto, onde dois processos se

comunicam trocando informações (Schepke et al, 2007). Também existem funções coletivas, onde vários processos estão envolvidos na comunicação, como a função *broadcast*, onde um processo envia dados para todos os outros processos ao mesmo tempo. Além disso, existem funções para o controle da aplicação, que informam o número de processos envolvidos em uma aplicação ou podem fazer a sincronização dos processos, por exemplo. Cada processo pode executar diferentes partes da aplicação, que deve ser especificado pelo programador. Em geral um programa em MPI possui a seguinte estrutura:

- Inicialização da comunicação e definição do número de processos;
- Cada processo executa e realiza as comunicações necessárias;
- Ao término da execução de todos os processos a comunicação é terminada.

Várias funções complexas estão disponíveis neste padrão de comunicação, mas qualquer aplicação pode ser desenvolvida utilizando as funções básicas de inicialização da comunicação, identificação de processos, envio e recebimento de mensagens e finalização da comunicação (Barros et al, 2013).

5.4 GPGPU e CUDA

Com o avanço de aplicações 3D, como jogos que exibem imagens cada vez mais próximas da realidade, foi necessário o avanço das unidades de processamento gráfico (GPUs). Elas precisam tratar todos os pixels de uma imagem em tempo real, possuindo então uma arquitetura diferente dos processadores usuais. As GPUs possuem vários processadores e estes possuem várias ULAs (unidade lógica aritmética), que são utilizadas para realizar uma mesma tarefa em vários pixels da imagem ao mesmo tempo. Observando este grau de paralelismo, as GPUs começaram a serem utilizadas na resolução de outros problemas, surgindo o que se denomina de GPGPU (General Purpose Graphics Processing Unit). A Figura 5.3 mostra uma comparação entre a arquitetura de uma CPU e uma GPU, onde pode ser visto que a GPU possui muito mais processadores com unidades de memória e controle simples.

O objetivo da computação em GPU é executar qualquer tipo de algoritmo, não necessariamente relacionado ao processamento gráfico. As GPUs possuem processadores

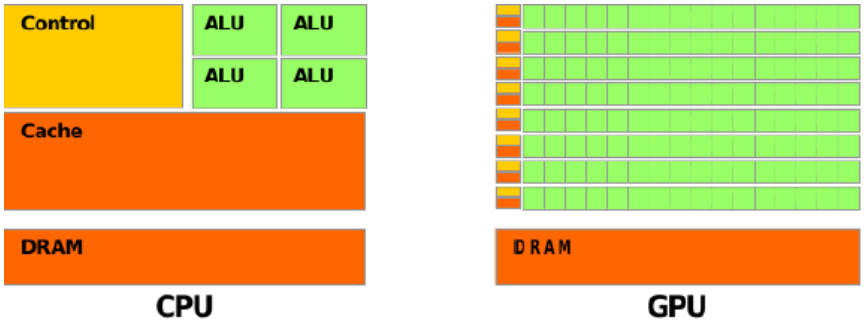


Figura 5.3: Comparação entre as arquiteturas da CPU e da GPU.

multicores altamente paralelos, *multithread* e com uma alta largura de banda de memória. Então programas de simulações computacionais, por exemplo, são escrito em uma linguagem que pode ser interpretada e executada pelo dispositivo, conseguindo assim um alto grau de paralelismo, e consequentemente um menor tempo de execução.

A plataforma CUDA, *Compute Unified Device Architecture*, é um modelo de computação paralela que foi desenvolvido pela NVIDIA (NVIDIA, 2013) com o objetivo de aproveitar o poder computacional das GPUs para resolver qualquer tipo de problema. Ela permite o uso da linguagem de programação C para executar códigos na GPU de modo transparente. O mesmo programa é executado em dados diferentes através da criação de *threads*. Devido a esta característica os problemas processados em GPUs devem ter grande intensidade aritmética para que se consiga um bom desempenho com a versão paralela.

A GPU pode ser vista como um co-processador massivamente paralelo voltado para o processamento de *threads*. Na arquitetura CUDA existem um processador, chamado de hospedeiro (*host*) e uma GPU, chamado de dispositivo (*device*). Então o processador pode executar uma parte do código e em seguida envia os dados para a GPU, através de uma operação de cópia da memória do processador para a memória da GPU. O dispositivo executa sua tarefa e ao fim envia os dados de volta para o processador, também através de cópia de memória.

Para se executar um código na GPU deve ser criada uma função chamada *kernel*, que é um tipo de função que é chamada pela CPU, mas é executada na GPU por várias *threads*. As *threads* são divididas em blocos e cada bloco é executado em um processador

da GPU. Um conjunto de blocos é chamado de *grid*, podendo ser de até três dimensões. Quando a CPU chama uma função *kernel*, ela deve especificar o tamanho do bloco e o tamanho do *grid*. Dentro do *kernel* existem funções que identificam cada bloco do grid e cada *thread* do bloco, onde cada *thread* geralmente se refere a um dado do problema. A memória é compartilhada entre as *threads* e se uma *thread* quer acessar uma posição de memória de uma outra *thread* que está no mesmo bloco, o acesso é mais rápido do que com *threads* de blocos diferentes.

Um código CUDA, em geral, possui a seguinte estrutura:

- Alocação de memória na CPU;
- Inicialização dos dados;
- Alocação de memória na GPU;
- Transferência dos dados da CPU para a GPU;
- Chamada à função *kernel* pela CPU;
- Execução do kernel na GPU;
- Transferência dos dados da GPU para a CPU;
- Libera memória da CPU e da GPU.

5.5 Técnicas aplicadas ao MLB

Neste trabalho foram desenvolvidas duas versões paralelas para diferentes problemas usando o MLB. O problema do escoamento de Poiseuille foi paralelizado com a técnica de memória distribuída, utilizando MPI, enquanto, o problema do Monodomínio foi aplicado a técnica de memória compartilhada através da arquitetura CUDA.

5.5.1 MPI

Na versão paralela em MPI o *lattice* foi dividido em blocos de linhas, de acordo com o número de processos envolvidos na computação, como mostra a Figura 5.4. Este particionamento foi feito devido a maior facilidade de implementação, apesar de não ser a

melhor estratégia quando a largura do domínio é muito maior do que a altura. Então as etapas de inicialização, colisão, propagação e tratamento das condições de contorno foram realizadas sobre os dados do *lattice* em paralelo, como mostra o Algoritmo 6. Além disso, como no problema do escoamento de Poiseuille, busca-se o estado estacionário, o cálculo do erro para a convergência ao estado estacionário também foi feito em paralelo.

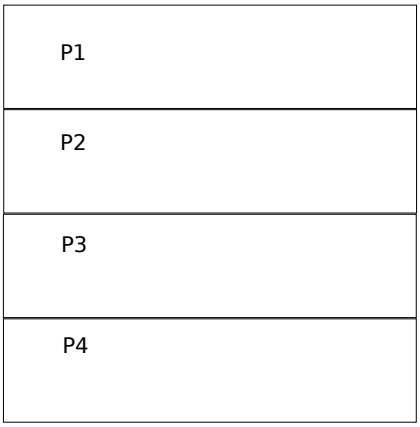


Figura 5.4: Exemplo de particionamento do *lattice* entre quatro processos

Foi visto no Algoritmo 4 que a etapa de propagação depende de dados de nós vizinhos a um determinado nó, como o *lattice* foi particionado e a implementação foi feita em memória distribuída, os processos precisarão de dados dos processos vizinhos. A Figura 5.5 ilustra a comunicação necessária entre os processos responsáveis por blocos vizinhos, onde as distribuições da borda do bloco P1 representadas por setas tracejadas precisam se propagar para o bloco P2 e vice-versa. Esta comunicação é feita através de funções de passagem de mensagem da biblioteca MPI.

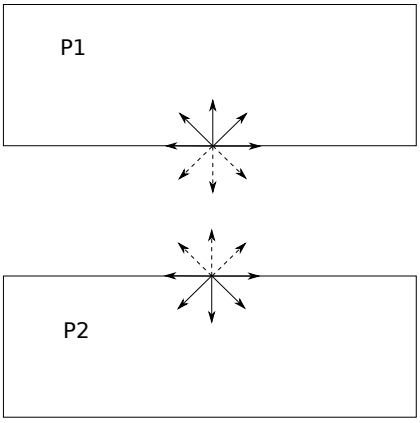


Figura 5.5: Comunicação entre processos

O Algoritmo 6 mostra a estrutura da implementação MPI para o escoamento

de Poiseuille utilizando o MLB. Primeiramente, cria-se os processos, onde o número de processos é dado pelo usuário. Como já foi falado o *lattice* foi dividido em blocos e cada processo é responsável pela computação de um bloco. As etapas de inicialização do *lattice* e colisão são totalmente locais, ou seja, o cálculo feito em cada nó só depende do valor deste mesmo nó no passo de tempo anterior. Então os processos realizam a inicialização de todos os blocos em paralelo e em seguida realizam a etapa de colisão para todos os blocos em paralelo. A comunicação necessária é feita utilizando as funções de envio e recebimento de mensagens do MPI e então a etapa de propagação pode ser realizada em paralelo. Após esta etapa acontece o cálculo do erro em paralelo, cada processo calcula uma parcela do erro e em seguida é realizada uma operação de redução, onde os valores calculados em cada processo são enviados e somados pelo processo principal. Para finalizar os dados que estão espalhados pelos processos são agrupados no processo principal, que os grava em arquivo para posterior visualização.

Algoritmo 6: Algoritmo paralelo do MLB para o problema de Poiseuille.

```
1  Configuração dos parâmetros do modelo
2  Criação dos processos
3  Cada processo inicializa as distribuições, a velocidade e densidade para cada
   nó do lattice que pertencem ao seu bloco
4  enquanto erro ≥ tolerancia faça
5      Cada processo realiza a etapa de colisão para seus dados
6      Processos responsáveis por blocos vizinhos trocam mensagem para
       receber os valores da borda
7      Cada processo realiza a etapa de propagação para seus dados
8      Cada processo realiza o tratamento para seus dados
9      Cada processo calcula uma parcela do erro entre a solução atual e a do
       passo de tempo anterior
10     Processo 0 junta todas as parcelas do erro
11     Processo 0 junta os blocos e grava resultado em arquivo
```

5.5.2 CUDA

Na implementação paralela utilizando CUDA, o *lattice* foi dividido em blocos como na Figura 5.6, onde cada bloco possui um determinado número de nós do *lattice*, 16×16, por exemplo. E ainda, o número de nós em cada bloco é igual ao número de *threads*, ou seja, cada *thread* será responsáveis pela computação de um nó. De forma similar à

implementação em MPI, as etapas de inicialização e colisão são totalmente locais, mas as etapas de propagação dependem de informações de nós vizinhos, entretanto neste caso a memória é compartilhada e não existe a necessidade de comunicação entre as *threads*. As *threads* podem acessar dados de outros blocos normalmente, apesar de o tempo de acesso a um endereço de memória de outro bloco ser mais lento.

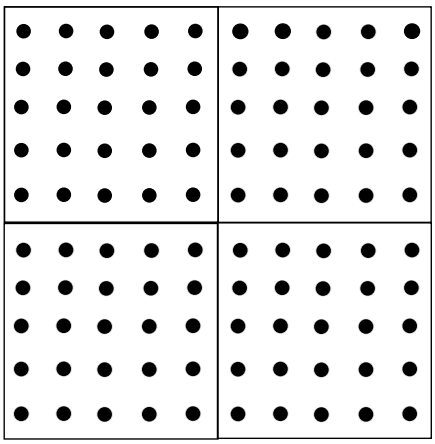


Figura 5.6: Exemplo de particionamento do *lattice* em CUDA

O Algoritmo 7 apresenta como foi feita a implementação em CUDA, onde foram criados quatro *kernels*: inicialização, colisão, propagação e contorno. Primeiramente é feita a alocação das variáveis na CPU e na GPU, em seguida é chamado o *kernel* de inicialização. A cada passo de tempo são chamados os *kernels* colisão, propagação e contorno. Além disso, em alguns passos de tempo os resultados são gravados em arquivo, para posterior visualização. Quem grava os dados é a CPU, então é necessário transferir os dados da GPU para a CPU.

Um exemplo de chamada à função *kernel* é apresentada no Algoritmo 8, onde é definido o número de *threads* por bloco e o tamanho do *grid*, em seguida o *kernel* é chamado informando o tamanho do bloco e do grid.

O Algoritmo 9 mostra como foi feita a implementação da colisão do MLB para o problema do Monodomínio. Primeiramente as coordenadas da *thread* são associadas ao nó do *lattice* que será tratado e cada *thread* é responsável por um nó. Cada distribuição do *lattice* é uma matriz, que por sua vez foi representada em um vetor contíguo na memória. Então é necessário fazer um mapeamento da coordenada (i, j) na coordenada k do vetor. Para diminuir o acesso à memória, as variáveis são lidas da memória e armazenadas em

Algoritmo 7: Algoritmo MLB em CUDA para problema da eletrofisiologia cardíaca.

```
1  Configuração dos parâmetros do modelo
2  Aloca variáveis na CPU com malloc
3  Aloca variáveis na GPU com cudaMalloc
4  Chama kernel de inicialização dos dados
5  enquanto tempo ≤ tempototal faça
6      Chama kernel da colisão
7      Chama kernel da propagação
8      Chama kernel do contorno
9      se t%5 = 0 então
10         Copia dados da GPU para CPU, com cudaMemcpyAsync e salva em
           arquivo
```

Algoritmo 8: Chamada ao *kernel* colisão

```
1  dim3 dimBlock(blocksizex, blocksizey)
2  dim3 dimGrid((nx + dimBlock.x - 1) / dimBlock.x, (ny + dimBlock.y - 1) /
   dimBlock.y )
3  colisao<<<dimGrid, dimBlock>>>(parametros)
```

variáveis locais. O potencial de ação é calculado no nó tratado e as variáveis do sistema de EDOs são atualizadas, através do método de Euler explícito, que está implementado na função *solveode*. Em seguida, calcula-se as distribuições de equilíbrio e aplica-se o operador de colisão.

A condição colocada no início do algoritmo é para tratar casos em que o número de *threads* maior do que o número de dados que serão tratados, por exemplo, se for definido um *lattice* de tamanho 20×20 e blocos de *threads* de 16×16, serão criados 4 blocos e algumas *threads* não serão mapeadas em dados.

No *kernel* da propagação, Algoritmo 10, o início é semelhante ao *kernel* da colisão. Em seguida são feitos testes para verificar se o nó atual é um nó do contorno e então é feita a propagação das distribuições.

A implementação paralela utilizando GPUs foi simples. Outras estratégias e otimizações (Myre et al, 2011; Bernaschi et al, 2010) ainda podem ser feitas com o intuito de diminuir ainda mais o tempo de simulação.

Algoritmo 9: *kernel* colisão para o problema do Monodomínio

```
1  i = blockIdx.x * blockDim.x + threadIdx.x
2  j = blockIdx.y * blockDim.y + threadIdx.y
3  if (i < nx  &&  j < ny) then
4      k = mapij(i,j)
5      f0l = devf0[k]
6      ⋮
7      f8l = devf8[k]
8      phil = f0l + f1l + f2l + f3l + f4l + f5l + f6l + f7l + f8l
9      feq0 = w0 * phil
10     ⋮
11     feq8 = w8 * phil
12     solveode(t, dt, devphi, neq*k)
13     devf0aux[k] = omega*feq0 + f0l*(1 − ω) + w0*I_ion
14     ⋮
15     devf8aux[k] = omega*feq8 + f8l*(1 − ω) + w8*I_ion
```

Algoritmo 10: *kernel* propagação para o problema do Monodomínio

```
1  i = blockIdx.x * blockDim.x + threadIdx.x
2  j = blockIdx.y * blockDim.y + threadIdx.y
3  if (i < nx  &&  j < ny) then
4      jm=(j==0) ? 0 : j-1
5      jp=(j==ny-1) ? ny-1 : j+1
6      im=(i==0) ? 0 : i-1
7      ip=(i==nx-1) ? nx-1 : i+1
8      //propagação:
9      k = mapij(i,j)
10     devf0[k]=devf0aux[k]
11     ⋮
12     devf8[k]=devf8aux[mapij(im, jp)]
```

6 Experimentos Computacionais

Neste capítulo serão apresentados os resultados dos experimentos realizados com os métodos LGAC e MLB. Primeiramente são apresentados os resultados das implementações do LGAC e MLB para problemas da dinâmica dos fluidos. Em seguida são apresentados os resultados encontrados com o MLB para o problema do Monodomínio. Finalizando o capítulo, é apresentado o desempenho alcançado com as versões paralelas do MLB utilizando MPI e CUDA.

6.1 LGAC - Modelo HPP

O primeiro experimento consistiu na dispersão de um gás em um compartimento fechado. O compartimento é dividido em duas partes por uma parede, sendo que uma parte da parede possui uma abertura que permite a passagem de partículas de um lado para outro. Uma configuração inicial aleatória do gás foi colocada de um lado do compartimento e este se propagou por todo ambiente, como pode ser visto na Figura 6.1, que mostra a densidade do gás. A condição de contorno reflexiva foi utilizada em todas as bordas e na parede dentro do domínio.

Um segundo experimento foi realizado, que consistiu na simulação de um compartimento fechado com uma configuração inicial aleatória para o gás. E ainda, uma região quadrada com alta densidade de gás no canto inferior esquerdo foi criada, como mostra a Figura 6.2. Pode-se observar nos dois experimentos que a solução possui ruídos, onde fica evidente os valores discretos assumidos por cada partícula. Para se obter uma solução suave para a densidade, pode-se considerar médias por amostras do domínio.

6.2 MLB para simulação de fluidos

Para avaliar a capacidade do MLB de simular problemas da dinâmica dos fluidos, foram implementados dois problemas clássicos desta área, que são considerados *benchmarks* da

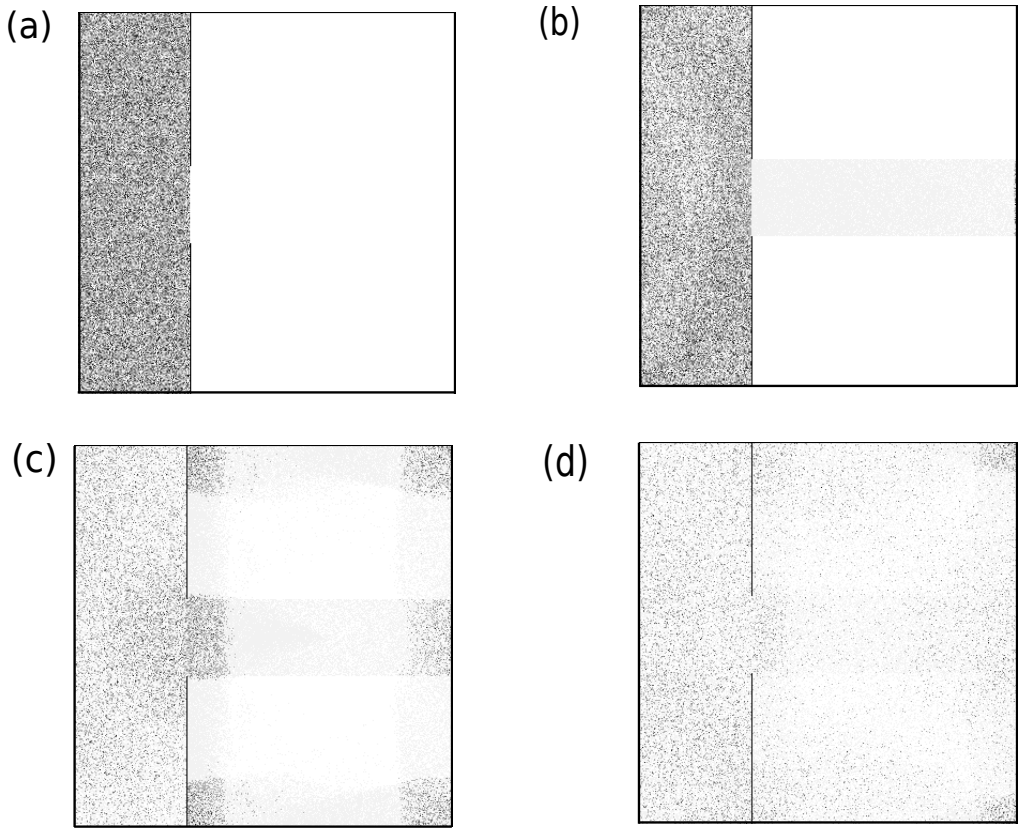


Figura 6.1: Densidade das partículas de um gás se dispersando pelo ambiente. Os pontos mais escuros possuem densidade maior. (a) Configuração inicial. (b), (c) e (d) mostram a dispersão do gás durante o tempo.

dinâmica dos fluidos.

6.2.1 Escoamento de Poiseuille

O escoamento de Poiseuille é um escoamento em regime estacionário de um fluido, por exemplo o de um liquido escoando dentro de um tubo. A velocidade no centro do tubo é máxima e vai diminuindo ao se afastar do centro até atingir velocidade zero na parede do tubo, formando um perfil de escoamento parabólico. Este problema possui solução exata e pode ser usado para se comparar com a solução aproximada do MLB. A solução exata

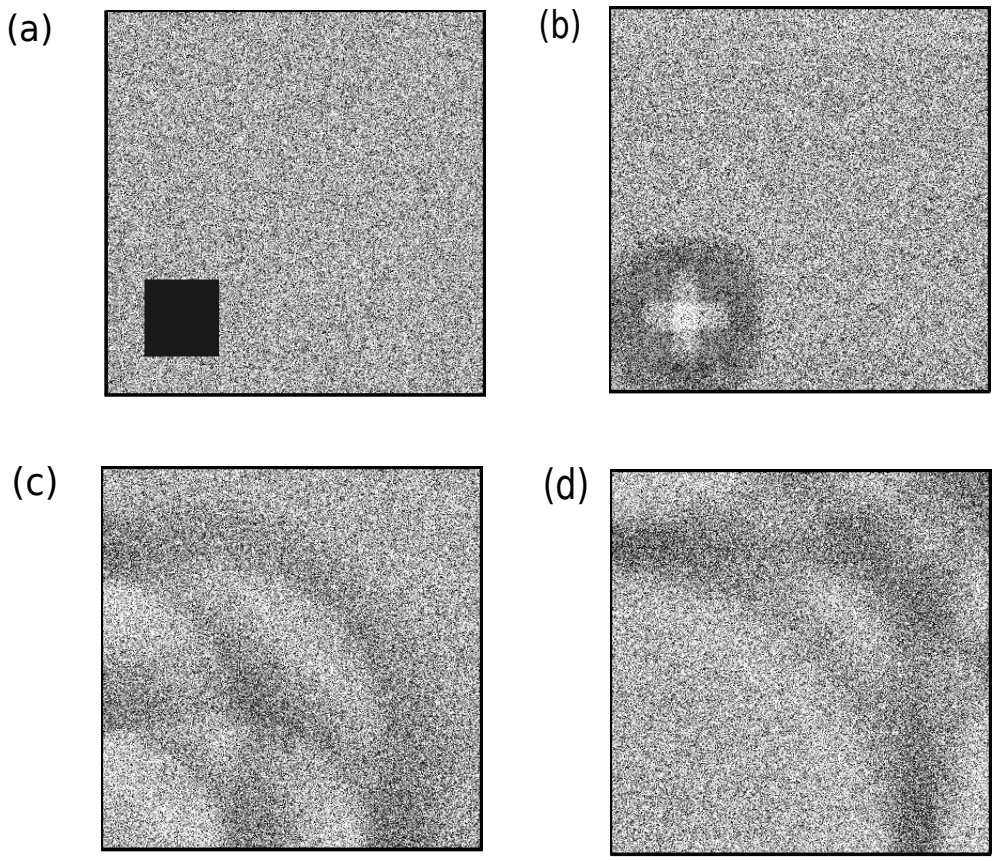


Figura 6.2: Densidade das partículas do fluxo de um gás em um espaço fechado. Os pontos mais escuros possuem densidade maior. (a) Configuração inicial. (b), (c) e (d) mostram a evolução do fluxo de gás pelo espaço fechado

para o problema bidimensional é dada pela seguinte equação:

$$u_x(x,y) = 4u_{max}\frac{y}{D}\left(1-\frac{y}{D}\right) \tag{6.1}$$

$$u_y(x,y) = 0 \tag{6.2}$$

onde u_{max} é a velocidade máxima dentro do canal, D é a altura do canal. Pode-se verificar que a velocidade varia parabolicamente na altura do canal. O escoamento de Poiseuille pode ser caracterizado pelo número adimensional de Reynolds, que indica o grau de turbulência do escoamento e é dado por:

$$Re = \frac{u_{max}D}{\nu} \tag{6.3}$$

onde ν é a viscosidade do fluido.

Este problema foi resolvido com o MLB utilizando o modelo D2Q9, para diferentes tamanhos de *lattice*. A Figura 6.3 mostra uma simulação usando um *lattice* de 64×32, com velocidade máxima $u_{max} = 1$, $Re = 10$ e altura do canal $D = 1$. Nas bordas superior e inferior foi utilizada a condição reflexiva, que implica em velocidade nula na parede, e nas bordas esquerda e direita foi utilizada a condição periódica. Além disso, para iniciar o escoamento foi utilizada uma força de corpo, onde suas componentes são:

$$G_x = \frac{8\rho u_{max}\nu}{6D^2} \tag{6.4}$$

$$G_y = 0 \tag{6.5}$$

onde ρ é a densidade do fluido. Então a equação do MLB fica da seguinte forma:

$$f_i(\mathbf{x} + \mathbf{e}_i\Delta x, t + \Delta t) - f_i(\mathbf{x}, t) = \frac{1}{\tau}(f_i(\mathbf{x}, t) - f_i^{eq}(\mathbf{x}, t)) + G_i \tag{6.6}$$

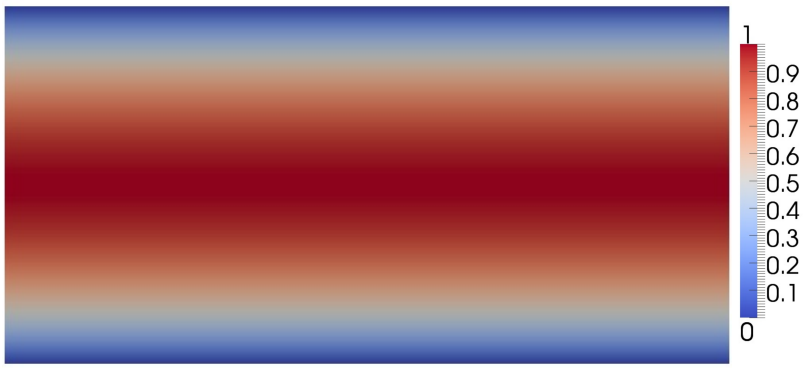


Figura 6.3: Simulação do escoamento de Poiseuille

Como o problema do escoamento de Poiseuille é um problema estacionário, e o MLB é naturalmente transiente, é preciso escolher um critério para verificar se o sistema chegou a um estado estacionário. Logo, com o objetivo de verificar a convergência do método para este problema, a distância entre dois campos vetoriais A e B, foi medida

usando a norma $L2$ relativa, da seguinte forma:

$$||\mathbf{A} - \mathbf{B}|| = \sqrt{\frac{\sum_{i=1}^{N}(\mathbf{A}_i - \mathbf{B}_i)^2}{\sum_{i=1}^{N}\mathbf{B}_i}} \tag{6.7}$$

onde $\mathbf{A}$ e $\mathbf{B}$ são os campos vetoriais e $\mathbf{A}_i$ e $\mathbf{B}_i$ são os vetores correspondentes a cada célula i dos campos $\mathbf{A}$ e $\mathbf{B}$. com todas as células do *lattice*. Para calcular a convergência para um estado estacionário $\mathbf{A}_i$ e $\mathbf{B}_i$ representam o campo de velocidades no tempo atual e anterior, respectivamente. Por outro lado, para calcular o erro usamos $\mathbf{A}$ como o campo de velocidade aproximado obtido pelo MLB e $\mathbf{B}$ como o valor da solução exata.

A tolerância adotada para se chegar ao estado estacionário foi de 10^{-8} e o erro entre a solução do MLB e a solução exata para um lattice de 64×32 foi de 1.925×10^{-3}. A Figura 6.4 mostra uma comparação gráfica entre a solução exata e a aproximada para este problema, onde estão sendo analisados os valores da velocidade em x ao longo de uma linha vertical do canal. E a Figura 6.5 mostra a convergência do MLB quando o

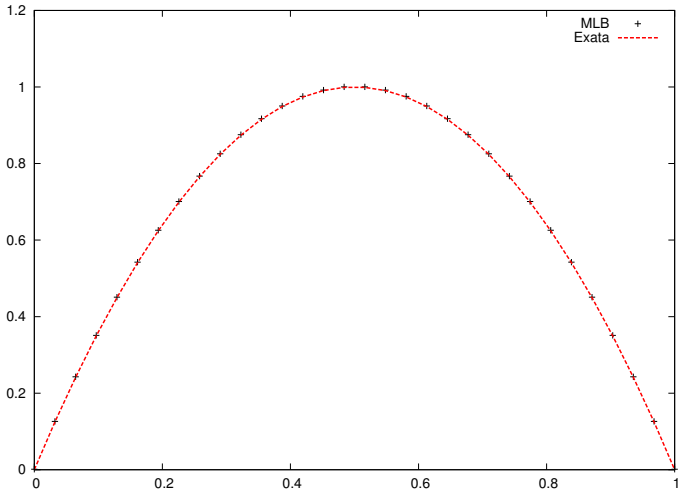


Figura 6.4: Comparação entre a solução do MLB e a exata para o escoamento de Poiseuille

lattice é refinado. Foram utilizados o mesmo passo de tempo e número de Reynolds para todos os tamanhos de *lattice*. O gráfico está em escala $log_2 \times log_2$ e pode-se observar que a convergência do MLB é quadrática para o tipo de condição de contorno utilizado.

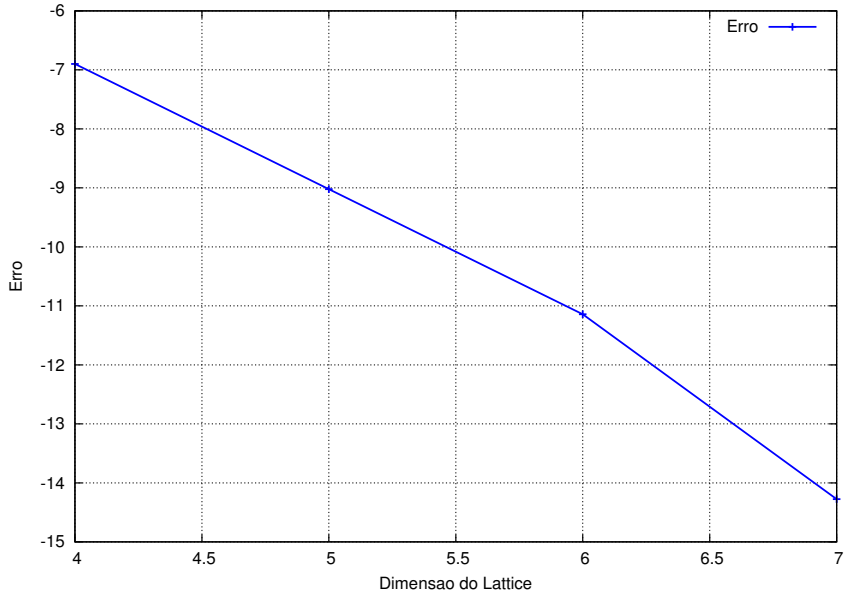


Figura 6.5: Convergência do MLB em escala $log_2 \times log_2$

6.2.2 Problema da cavidade

Outro problema muito utilizado para testar simulações da dinâmica dos fluidos é o problema da cavidade quadrada, onde uma parede se desloca com velocidade horizontal enquanto as outras três paredes permanecem em repouso, como pode ser visto na Figura 6.6. O movimento dentro do domínio é dado pela difusão de efeito viscosos causados pelo movimento da parede superior. Para as paredes em repouso foi utilizada a condição de contorno reflexiva e na parede em movimento foi utilizada a condição de contorno de velocidade prescrita, descritas na Seção 3.4.3.

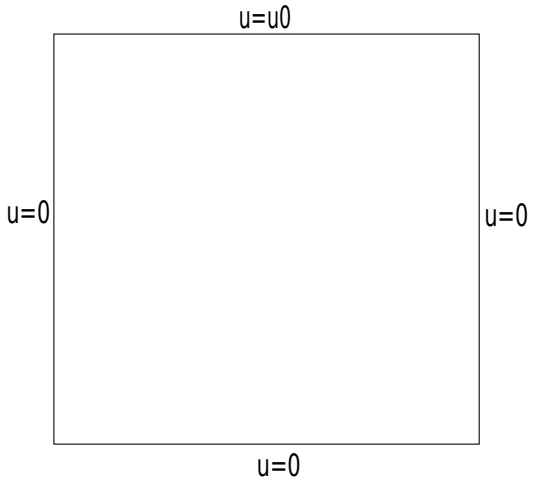
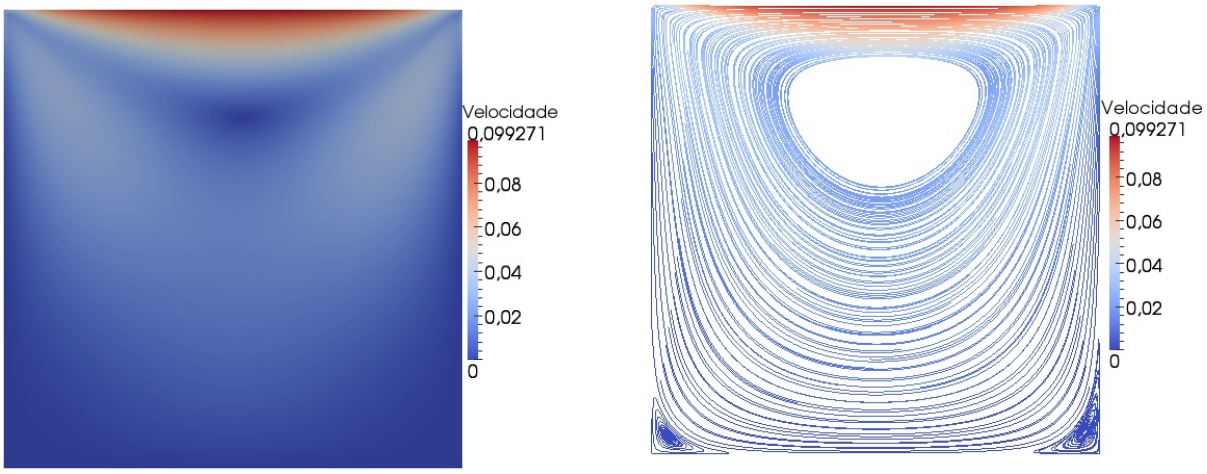


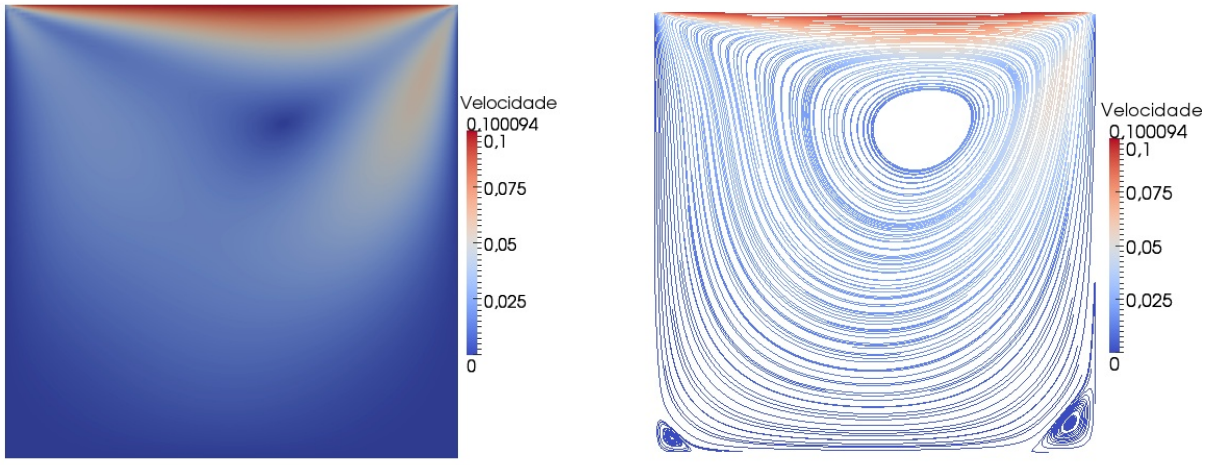
Figura 6.6: Condições de contorno do problema da cavidade.

Nesta simulação foi utilizado um *lattice* de dimensão 100×100, com número de Reynolds $Re = 1000$ Figura 6.9, $Re = 100$ Figura 6.8 e $Re = 10$ Figura 6.7. As figuras mostram o campo de velocidade e as linhas de corrente da cavidade quadrada, para os diferentes números de Reynolds.



(a) (b)

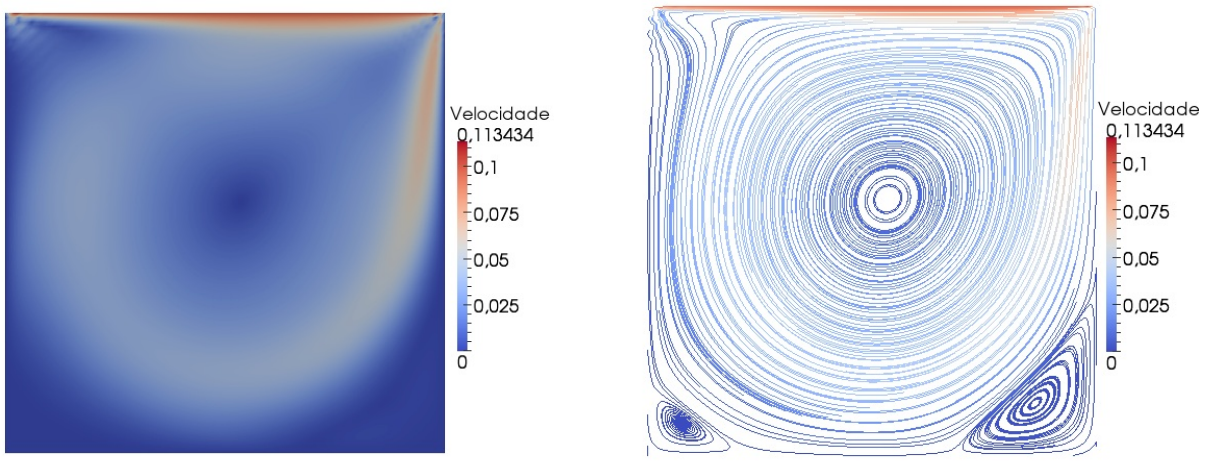
Figura 6.7: Cavidade quadrada Re=10. (a) Magnitude do campo de velocidades da cavidade. (b) Linhas de corrente.



(a) (b)

Figura 6.8: Cavidade quadrada Re=100. (a) Magnitude do campo de velocidades da cavidade. (b) Linhas de corrente.

O problema da cavidade quadrada não tem solução analítica, então a solução obtida com o MLB foi comparada com a solução encontrada por (Ghia et al, 1982) para o mesmo problema através do método dos elementos finitos. A Figura 6.10 apresenta uma comparação gráfica entre as duas soluções, onde foram avaliados valores de uma coluna e de uma linha no centro da cavidade.



(a) (b)

Figura 6.9: Cavidade quadrada Re=1000. (a) Magnitude do campo de velocidades da cavidade. (b) Linhas de corrente.

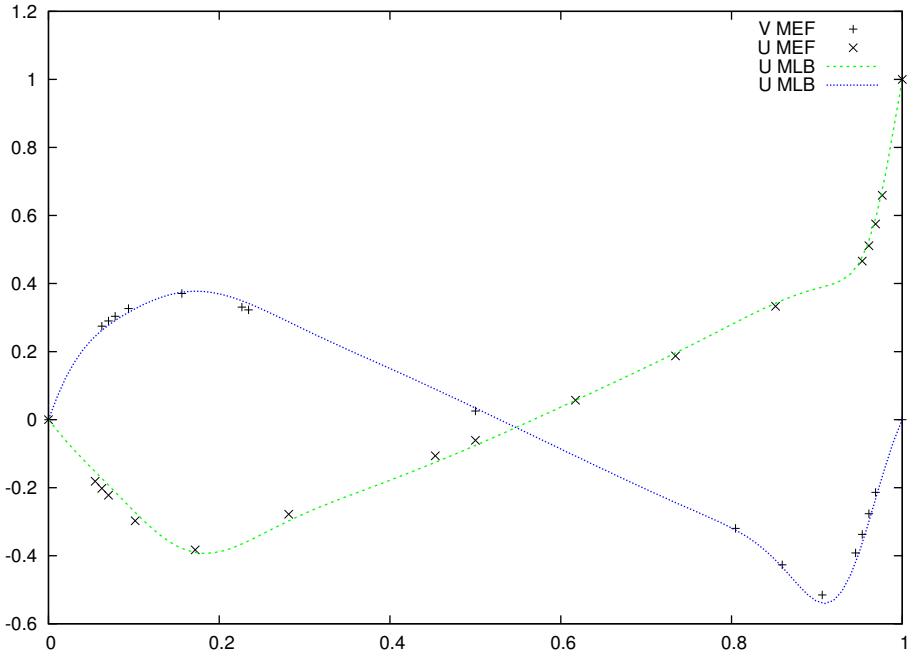


Figura 6.10: Comparação entre a solução do MLB e a solução com MEF para Re=1000.

6.3 MLB para simulação de problemas de reação-difusão

Para demonstrar a aplicação do método de lattice Boltzmann para a solução numérica do modelo do monodomínio, descrito no Capítulo 4, dois cenários diferentes foram escolhidos para simulação, um utilizando o modelo celular de Mitchell-Schaeffer e outro o modelo de Luo-Rudy. Para ambos os casos apresentamos também resultados numéricos obtidos em uma implementação computacional paralela explorando o poder computacional das modernas placas gráficas através da arquitetura NVIDIA CUDA.

A Figura 6.11 mostra um experimento realizado com o modelo do monodomínio acoplado ao modelo Mitchell-Schaeffer e representa uma situação com uma onda de reentrada, em geral, associada a arritmias cardíacas. Para iniciar essa simulação, inicialmente um primeiro estímulo é aplicado em uma parte do tecido gerando a propagação de uma onda plana, como mostra a Fig. 6.11(a). Depois de alguns instantes, um segundo estímulo é aplicado em outra parte do tecido (Fig. 6.11(b)), o que inicia a formação de uma onda espiral, a qual é auto-sustentável e se mantém durante toda a simulação, indicando uma situação de comportamento arrítmico do coração. Para o modelo celular de Luo-Rudy as simulações consistiram apenas na simulação da propagação de uma onda plana, como a da Figura 6.11(a).

Os experimentos computacionais consistiram na simulação de uma parte do tecido cardíaco com 4 cm de largura e 4 cm de comprimento. Para a discretização temporal foi utilizado um passo de tempo de $\Delta t = 0.1\ ms$ para o modelo MS e de $\Delta t = 0.01\ ms$ para o modelo LR, enquanto que para a discretização espacial um espaçamento de $\Delta x = 0.02\ cm$ foi utilizado. Para representar a condutividade isotrópica, o seguinte valor $\sigma = 0.0003\ cm^2/ms$, foi adotado (Rapaka et al, 2012). A atividade elétrica do coração foi simulada por um segundo e foram realizadas tanto com uma versão sequencial do programa quanto com uma implementação paralela do problema utilizando CUDA.

6.4 Implementações paralelas

Com o objetivo de avaliar o desempenho do método de lattice Boltzmann em ambientes de computação paralela, foram desenvolvidas duas versões paralelas para diferentes problemas. A primeira versão foi desenvolvida utilizando uma biblioteca MPI para o problema do escoamento de Poiseuille e a segunda versão foi implementada em CUDA para o problema do Monodomínio. A estratégia de paralelização nos dois casos foi a de particionamento dos dados, onde processos/*threads* executam o mesmo código em dados diferentes. Mais detalhes sobre as técnicas utilizadas podem ser vistos na Seção 5.5.

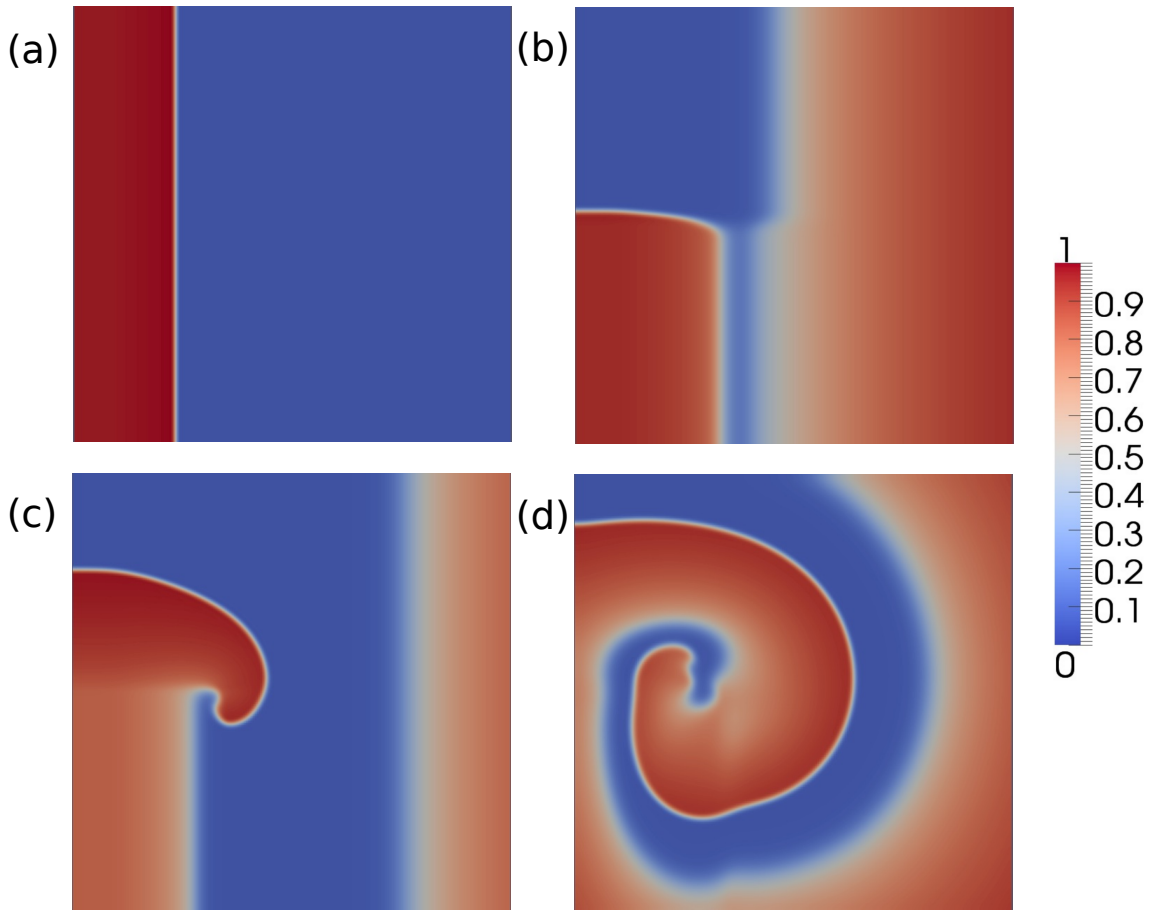


Figura 6.11: Exemplo da distribuição espacial do potencial transmembrânico na simulação do modelo Monodomínio com o modelo celular Mitchell-Schaeffer usando o método de Lattice Boltzmann e Euler explícito durante 1 s (a) Primeiro estímulo aplicado (b) Segundo estímulo aplicado (c) Formação de uma espiral (d) Espiral que mostra comportamento arrítmico do coração.

6.4.1 MPI

Os resultados apresentados foram obtidos as máquinas do cluster do Programa de Pós Graduação de Modelagem Computacional, com máquinas com 12 GB de memória RAM e um processador Intel Xeon E5620 de 2.40GHz com 4 cores e 12MB de memória cache. O código foi compilado utilizando a biblioteca MPICH e a versão 4.1.2 do compilador GCC sem o uso de *flags* de otimização. Usando este ambiente as simulações foram realizadas em um Linux 2.6.18 x86-64 usando C++ com um core e foram comparadas com as simulações executadas para diferentes números de processadores. Tentou-se alocar o maior número de processos em máquinas diferentes, com o objetivo de analisar se a comunicação entre processos afetaria o desempenho da implementação.

Antes de realizar a implementação paralela, a primeira tarefa realizada foi a veri-

ficação da porcentagem do código sequencial que poderia ser paralelizado, com o intuito de descobrir qual a aceleração máxima que se pode alcançar com uma implementação paralela. E o resultado obtido aplicando a Lei de Amdahl ao problema, pode ser visto na Tabela 6.1, onde foi considerado que as etapas de inicialização, colisão, propagação e contorno podem ser paralelizadas. A parte do código que não foi paralelizada está relacionada com operações de entrada e saída.

Processadores	256×128	512×256	1024×512
2	1.920	1.988	1.990
4	3.556	3.932	3.950
8	6.195	7.693	7.753
16	9.851	14.74	14.97
32	13.97	27.20	28.04

Tabela 6.1: *Speedup* pela Lei de Amdahl, para o problema do escoamento de Poiseuille

Foi realizada a implementação paralela e em seguida foram feitos testes com os diferentes tamanhos de *lattice* e número de processos, tanto para a versão paralela quanto para a versão sequencial e os resultados são apresentados a seguir. A Tabela 6.3 mostra o *speedup* calculado com os tempos de execução obtidos, que podem ser vistos na Tabela 6.2. Todas as medidas de tempo apresentadas nesta seção foram realizadas cinco vezes

Processadores	256×128	512×256	1024×512
Sequencial	68.476	366.992	1633.494
2	38.200	167.138	832.172
4	21.364	88.962	357.268
8	14.896	47.062	186.720
16	12.228	27.976	102.666
32	11.590	20.558	52.588

Tabela 6.2: Tempo de execução medido em segundos, medido com o comando time do sistema operacional Linux

e em seguida calculada a média aritmética, para cada tamanho de *lattice* e número de processadores. O desvio padrão relativo máximo foi de 5%.

Nota-se que com o menor tamanho de *lattice* o *speedup* foi pior, mas fica perto do *speedup* dado pela Lei de Amdahl, quando o número de processos é pequeno. Já com o aumento do número de processos, o *speedup* cai devido ao aumento de comunicação. Aumentando o tamanho do *lattice* é possível observar que em alguns casos o *speedup* chega a ser super-linear. O código começa a ter uma intensidade aritmética maior e consequen-

Processadores	256×128	512×256	1024×512
2	1.792	2.195	1.962
4	3.205	4.125	4.572
8	4.596	7.798	8.748
16	5.599	13.11	15.91
32	5.908	17.85	**31.06**

Tabela 6.3: *Speedup* obtido

temente, a parte sequencial fica menos significativa, outra explicação para este resultado pode ser devido à grande porcentagem de acerto de cache, já que o dado está dividido entre os processos e pode ser armazenado dentro da cache de cada processador. Além disso, a versão sequencial não foi compilada com *flags* de otimização, estas otimizações feitas pelo compilador podem diminuir o tempo de execução sequencial e talvez a aceleração alcançada deixe de ser super-linear.

A Figura 6.12 mostra o *speedup* obtido e seu comportamento super-linear, variando o número de processos e o tamanho do *lattice*. Pode-se observar que dobrando o número de processos e o tamanho do *lattice*, o *speedup* permanece próximo do valor anterior ou aumenta. Mas quando fixa-se um tamanho de *lattice* e aumenta-se o número de processos, o *speedup* diminui.

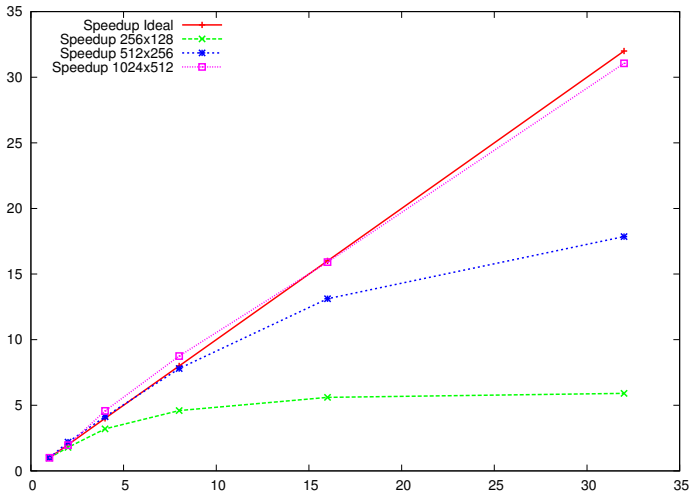


Figura 6.12: *Speedup* do MLB aplicado ao problema do escoamento de Poiseuille.

A Tabela 6.4 mostra a eficiência do código MPI, para os diferentes números de processos e tamanho de *lattice*.

Pode-se notar que o algoritmo é escalável, já que a eficiência diminui com o aumento do número de processadores e se mantém ou melhora quando o tamanho do

Processadores	256×128	512×256	1024×512
2	0.8962	1.0978	0.9814
4	0.8013	1.0313	1.1430
8	0.5746	0.9747	1.0935
16	0.3499	0.8198	0.9944
32	0.1846	0.5578	0.9706

Tabela 6.4: Eficiência do algoritmo paralelo

lattice e o número de processos dobra. A Figura 6.13 mostra o gráfico da eficiência do algoritmo implementado com MPI.

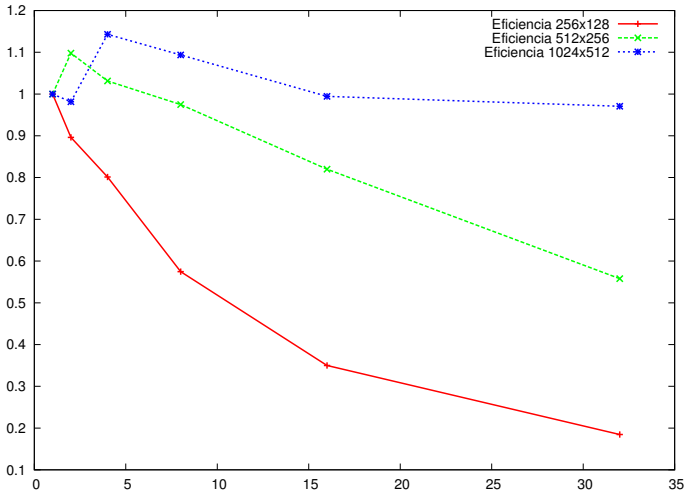


Figura 6.13: Eficiência

6.4.2 CUDA

Foi realizada uma implementação em CUDA do MLB para o problema do Monodomínio, com o objetivo de diminuir seu tempo de execução. É importante ressaltar aqui, que existem diversas possibilidades para se otimizar o código do MLB tanto na CPU quanto para a sua execução na GPU usando CUDA (Myre et al, 2011; Bernaschi et al, 2010). Entretanto, as implementações apresentadas nesse trabalho são simples e não foram otimizadas com o intuito de se obter o melhor desempenho possível. O objetivo aqui é demonstrar a aplicação do método e assim as possibilidades de se obter bons resultados em termos de tempo de execução para simulação da eletrofisiologia cardíaca mesmo com uma implementação paralela simples.

Os resultados apresentados foram obtidos utilizando uma máquina com 12 GB de memória RAM e um processador Intel Xeon E5620 de 2.40GHz com 4 cores e 12MB

de memória cache. Além disso a máquina possui uma unidade de processamento gráfico GPU NVIDIA Tesla M2050 com 14 multiprocessadores, totalizando 448 cores de 1.15GHz e 3GB de memória. O código foi compilado utilizando a versão 4.2 da arquitetura NVIDIA CUDA e a versão 4.1.2 do compilador GCC com o uso da flag de otimização `-O3`. Usando este ambiente as simulações foram realizadas em um Linux 2.6.18 x86-64 usando C++ com um core comparado com as simulações executadas na GPU.

A Tabela 6.5 apresenta o tempo de execução da versão serial da implementação numérica, utilizando o modelo celular de Mitchell-Schaeffer, assim como o tempo de execução da versão paralela, além da aceleração obtida com a implementação paralela. O programa foi executado para diferentes tamanhos de malha. A versão paralela o tempo de simulação com a malha de 200×200 foi muito próximo ao tempo real, isto é, para simular 1 s de atividade elétrica o tempo de execução do programa foi de 5 s. Já com uma malha mais refinada foi alcançada uma aceleração maior.

Na Tabela 6.6 é apresentado o tempo de execução das implementações numéricas serial e paralela para o modelo celular de Luo-Rudy. Assim como a aceleração alcançada com o algoritmo paralelo. Com este segundo modelo celular foi alcançado uma aceleração bem maior para os mesmos tamanho de malha, apesar do maior tempo de execução. Este maior tempo gasto para encontrar a solução é devido ao maior grau de complexidade do modelo e consequentemente um maior número de operações realizadas.

Tamanho da malha	Tempo Sequencial	Tempo Paralelo	Aceleração
200×200	98.89	5.37	18.42
400×400	407.47	9.66	42.17
800×800	2022.29	27.02	74.84

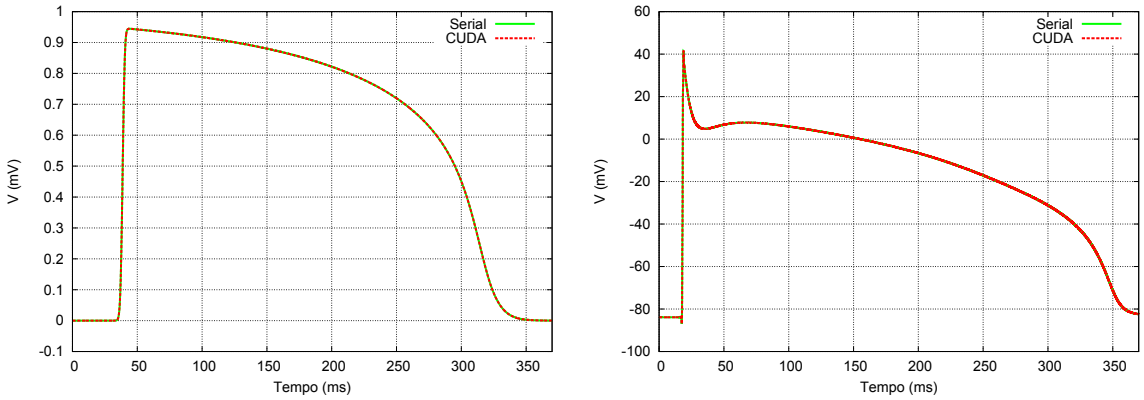
Tabela 6.5: Tempo de execução em segundos e aceleração obtida para o modelo MS.

Tamanho da malha	Tempo Sequencial	Tempo Paralelo	Aceleração
200×200	6947.02	47.69	145.67
400×400	28039.02	171.79	163.22
800×800	87049.09	455.22	191.22

Tabela 6.6: Tempo de execução em segundos e aceleração obtida para o modelo LR.

Outro experimento realizado foi a comparação entre a precisão da solução encontrada com o algoritmo sequencial e sua versão paralela. Foi observado o potencial elétrico

de uma célula do tecido durante uma simulação de 500 ms tanto na implementação sequencial quanto na versão paralela. A comparação gráfica entre as duas soluções para um nó no meio do tecido ao longo do tempo, para os dois modelos celulares apresentados, pode ser vista na Figura 6.14. O erro entre as duas soluções foi calculado utilizando a



(a) (b)

Figura 6.14: Comparação gráfica entre o potencial de ação da solução do modelo do monodomínio em um ponto da malha da implementação serial e paralela na GPU, (a) para o modelo celular Mitchell-Schaeffer e (b) para o modelo celular Luo-Rudy.

seguinte equação:

$$erro = \frac{\sqrt{\sum_{i=1}^{nt}\sum_{j=1}^{nv}(V(i,j) - V_{ref}(i,j))^2}}{\sqrt{\sum_{i=1}^{nt}\sum_{j=1}^{nv}V_{ref}(i,j)^2}}, \tag{6.8}$$

e foi observado que para o modelo MS o erro foi de 5.07×10^{-7} e para o modelo LR o erro foi de 1.06×10^{-4}. Os valores obtidos para o erro indicam que a implementação paralela é viável considerando que o tempo gasto na solução é muito menor e que ainda assim possuem uma boa precisão numérica em comparação com a implementação sequencial (CPU).

7 Conclusão

Com este trabalho espera-se mostrar que o métodos *Lattice-Gas* Autômato Celular e o método de lattice Boltzmann podem ser usados para a simulação de escoamento de fluidos assim como para problemas de reação-difusão.

Para simular o escoamento de um gás em um reservatório usando um LGAC o modelo HPP foi implementado e dois experimentos simples foram realizados. Os resultados foram avaliados de forma qualitativa e mostram que o escoamento de fluidos podem ser simulados usando LGAC. Modelos mais complexos como o FHP não foram testados, mas sabe-se que produzem resultados mais realísticos.

O MLB foi utilizado para resolver os dois problemas da dinâmica dos fluidos computacional, para diferentes tamanhos de *lattice* e número de Reynolds. Foi verificado que o erro numérico no caso do escoamento de Poiseuille, caso que possui solução analítica, e mostrou-se que o MLB se aproxima de tal solução de maneira satisfatória. No caso do problema da cavidade quadrada, que não possui solução analítica, o resultado do MLB foi comparado com uma solução obtida pelo MEF e o método de lattice Boltzmann ficou bem próximo da solução encontrada pelo método dos elementos finitos. Isto mostra que o MLB é capaz de resolver diferentes problemas da dinâmica dos fluidos. E quando submetido ao ambiente de computação paralela, o MLB obteve um bom desempenho na comparação entre o código sequencial e paralelo.

Também foi desenvolvido e implementado o método de lattice Boltzmann para a solução numérica de modelos matemáticos do tipo reação-difusão que descrevem o comportamento da eletrofisiologia cardíaca. Nesse caso ainda foi demonstrado o poder computacional das placas de processamento gráfico modernas, quando utilizadas para processamento científico.

A implementação do método obteve resultados promissores, mostrando a capacidade do método de resolver problemas de reação-difusão como o presente problema da eletrofisiologia cardíaca. O método oferece facilidade para implementações paralelas, que diminui muito o tempo de execução da simulação, ficando próximo ao tempo real em

algumas situações. Dois diferentes modelos celulares foram utilizados para a resolução do modelo monodomínio. O modelo mais simples, quando executado em paralelo fica mais próximo ao tempo real do problema. Já o modelo mais complexo possui um tempo de execução maior, mas o ganho com a implementação paralela é bem maior, onde é possível observar uma redução de mais de 190 vezes no tempo de simulação. Além disso, a precisão numérica foi mantida com a versão paralela.

Apesar do ganho computacional, a implementação paralela utilizando GPUs foi simples. Outras estratégias e otimizações (Myre et al, 2011; Bernaschi et al, 2010) ainda podem ser feitas com o intuito de diminuir ainda mais o tempo de simulação, as quais serão objeto de análise para trabalhos futuros, assim como a implementação do MLB para simulações computacionais 3D envolvendo geometrias complexas como a do ventrículo esquerdo.

Os métodos podem ser usados para a simulação complexas da dinâmica dos fluidos, com diferentes números de Reynolds, que indica se um escoamento é turbulento ou laminar. Eles aceitam diferentes condições de contorno e diferentes geometrias do domínio simulado, o que torna o método flexível para diferentes aplicações. Além disso, o método MLB não se limita a problemas de dinâmica dos fluidos. Ele pode ser usado para simular problemas de difusão, como problemas de transferência de calor, reação-difusão, que é usado para simular a eletrofisiologia do coração, por exemplo. Ou ainda problemas de meios porosos, como a simulação de reservatórios de petróleo. E ainda se comporta de maneira satisfatória quando submetido a ambientes de computação paralela, obtendo-se simulações próximas ao tempo real.

7.1 Trabalhos Futuros

Como trabalhos futuros pretende-se realizar simulações tridimensionais, utilizando o método de lattice Boltzmann para o problema de reação-difusão submetido a geometrias irregulares, como a geometria do coração. A realização de simulações 3D para os problemas da dinâmica dos fluidos também seriam interessantes, além da aplicação do método ao problema de meios porosos, que também é de grande interesse científico, como na simulação de reservatórios de petróleo.

Este trabalho não apresentou a análise multiescala de Chapman-Enskog, onde é possível mostrar que a equação do MLB recupera o comportamento macroscópico das equações de Navier-Stokes (ou das equações de reação-difusão aqui estudadas). Em trabalhos futuros pretende-se estudar a teoria da análise em multiescala para se compreender melhor o MLB, realizar uma análise de erro mais detalhada, assim como realizar comparações e validações com implementações que usam outros métodos numéricos.

A implementação paralela em CUDA foi realizada de forma simples, sem nenhuma otimização. Várias otimizações podem ser feitas para diminuir ainda mais o tempo de execução MLB, por exemplo, utilizando a memória de textura da GPU ou ainda fazendo o uso de mais de uma GPU para realizar a computação.

Referências Bibliográficas

de Barros, B. G. **Simulações computacionais de arritmias cardíacas em ambientes de computação de alto desempenho do tipo multi-gpu**. 2013. Dissertação de Mestrado - Universidade Federal de Juiz de Fora. Programa de Pós Graduação em Modelagem Computacional.

Beaumont, J.; Davidenko, N.; Davidenko, J. M. ; Jalife, J. Spiral waves in two-dimensional models of ventricular muscle: formation of a stationary core. **Biophys J**, v.75, p. 1–14, 1998.

Bernaschi, M.; Fatica, M.; Melchionna, S.; Succi, S. ; Kaxiras, E. A flexible high-performance lattice boltzmann gpu code for the simulations of fluid flows in complex geometries. **Concurre**, v.22, n.1, p. 1–14, 2010.

Bhatnagar, P. L.; Gross, E. P. ; Krook, M. A model for collisional processes in gases i: small amplitude processes in charged and in neutral one-component systems. **Phys Rev**, v.94, p. 511, 1954.

Blaak, R.; Sloot, P. M. A. Lattice dependence of reaction-diffusion in lattice boltzmann modeling. **Computer Physics Communications**, v.129, p. 256–266, 2000.

Dawnson, S. P.; Chen, S. ; D., D. G. Lattice boltzmann computations for reaction-diffusion equations. **J Chem Phys**, v.98, p. 1514–1523, 1992.

Fortuna, O. A. **Técnicas Computacionais para Dinâmica dos Fluidos. Conceitos Básicos e Aplicações.** 1. ed., São Paulo: Edusp, 2000.

Franco, N. B. **Cálculo Numérico**. 1. ed., São Paulo: Pearson, 2006.

Frisch, U.; Hasslacher, B. ; Pomeau., Y. Lattice gas cellular automata for the navier-stokes equations. **Phys. Rev. Lett.**, v.56, p. 1505, 1986.

Ghia, U.; Ghia, K. N. ; Shin, C. T. High-re solutions for incompressible flow using the navier-stokes equations and a multgrid method. **Journal of Computational Fhysics**, v.48, p. 387–411, 1982.

Golbert, D. R.; Blanco, P. J. ; Feijoo, R. A. **Simulation of 2d and 3d incompressible fluid flows via a lattice-boltzmann model**. In: International Conference on Particle-Based Methods Particles, Barcelona, 2009. CIMNE.

Golbert, D. R. **Modelos de lattice-boltzmann aplicados à simulação computacional do escoamento de fluidos incompressíveis.** 2009. Dissertação de Mestrado - Laboratório Nacional de Computação Científica. Programa de Pós Graduação em Modelagem Computacional.

Groop, W.; Lusk, E. **User's guide for mpich, a portable implementation of MPI.** Technical report, Argonne National Laboratory, 1994.

He, X.; Luo, L. S. Theory of the lattice boltzmann method: from the boltzmann equation to the lattice boltzmann equation. **Physical Review E**, v.56, p. 6811 – 6817, 1997.

Hodgkin, A.; Huxley, A. A quantitative description of membrane current and its application to conduction and excitation in nerve. **Journal of Physiology**, v.117, p. 500–544, 1952.

Hughes, T. J. R. **The Finite Element Method: Linear Static and Dynamic Finite Element Analysis (Dover Civil and Mechanical Engineering)**. 1. ed., N.J.: Prentice Hall, 2000.

LeVeque, R. J. **Finite Volume Methods for Hyperbolic Problems (Cambridge Texts in Applied Mathematics)**. 1. ed., Cambridge: Cambridge University Press, 2002.

LeVeque, R. J. **Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time-Dependent Problems (Classics in Applied Mathematic)**. 1. ed., Philadelphia: Siam, 2007.

Luo, C.; Rudy, Y. A model of the ventricular cardiac action potential. depolarization, repolarization, and their interaction. **Circ Res**, v.68, n.6, p. 1501–26, 1991.

Mitchell, C. C.; Schaeffer, D. G. A two-current model for the dynamics of cardiac membrane. **Bulletin of Mathematical Biology**, v.65, p. 767–793, 2003.

Mohamad, A. A. **Lattice Boltzmann Method: Fundamentals and Engineering Applications with Computer Codes.** 1. ed., Calgary: Springer, 2011.

Myre, J.; Walsh, S. D. C.; Lilja, D. ; Saar, M. O. Performance analysis of single-phase, multiphase, and multicomponent lattice-bolzmann fluid flow simulations on gpu clusters. **Concur**, v.23, p. 332–350, 2011.

NVIDIA. **Cuda**. Publicação Eletrońica: https://developer.nvidia.com/get-started-cuda-cc, Julho 2013.

Nash, M. P.; Panfilov, A. V. Electromechanical model of excitable tissue to study reentrant cardiac arrhythmias. **Progress in Biophysics and Molecular Biology**, v.85, p. 501–522, 2004.

Pacheco, P. S. **A Introduction to Parallel Programming**. 1. ed., San Francisco: Elsevier, 2011.

Plonsey, R. Bioelectric sources arising in excitable fibers (ALZA lecture). **Ann Biomed Eng**, v.16, n.6, p. 519–46, 1988.

Quinn, M. J. **Parallel Programming in C with MPI and OpenMP**. McGraw-Hill Education Group, 2003.

Rapaka, S.; Mansi, T.; Georgescu, B.; Pop, M.; Wright, G. A.; Kamen, A. ; Comaniciu, D. **Lbm-ep: Lattice-boltzmann method for fast cardiac electrophysiology simulation from 3d images**. In: Lecture Notes in Computer Science, MICCAI, 2012.

SCHEPKE, C. **Distribuição de dados para implementações paralelas do método de lattice boltzmann**. 2007. Dissertação de Mestrado - UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL INSTITUTO DE INFORMÁTICA. Programa de Pós Graduação em Computação.

Sundnes, J. **Computing the electrical activity in the heart**. Springer Verlag, 2006.

Wolf-Gladrow, D. A. **Lattice-Gas Cellular Automata and Lattice Boltzmann Models: An Introduction (Lecture Notes in Mathematics).** 1. ed., Bremerhaven: Springer, 2000.

Zou, Q.; He., X. On pressure and velocity boundary conditions for the lattice boltzmann bgk model. **American Institute of Physics**, v.9, p. 1591–1598, 1997.