Joventino de Oliveira Campos

Método de *lattice* Boltzmann para simulação da eletrofisiologia cardíaca em paralelo usando GPU

Dissertação apresentada ao Programa de Pós-graduação em Modelagem Computacional, da Universidade Federal de Juiz de Fora como requisito parcial à obtenção do grau de Mestre em Modelagem Computacional.

Orientador: Prof. D.Sc. Bernardo Martins Rocha

Coorientador: Prof. D.Sc. Rodrigo Weber dos Santos

Juiz de Fora

2015

Ficha catalográfica elaborada através do programa de geração
automática da Biblioteca Universitária da UFJF,
com os dados fornecidos pelo(a) autor(a)

Campos, Joventino de Oliveira.
Método de lattice Boltzmann para simulação da eletrofisiologia cardíaca em paralelo usando GPU / Joventino de Oliveira Campos. -- 2015.
104 p. : il.

Orientador: Bernardo Martins Rocha
Coorientador: Rodrigo Weber Dos Santos
Dissertação (mestrado acadêmico) - Universidade Federal de Juiz de Fora, ICE/Engenharia. Programa de Pós-Graduação em Modelagem Computacional, 2015.

1. Método de lattice Boltzmann. 2. Eletrofisiologia cardíaca. 3. Monodomínio. 4. Computaçao de alto desempenho. I. Rocha, Bernardo Martins, orient. II. Dos Santos, Rodrigo Weber, coorient. III. Título.

Joventino de Oliveira Campos

Método de *lattice* Boltzmann para simulação da eletrofisiologia cardíaca em paralelo usando GPU

Dissertação apresentada ao Programa de Pós-graduação em Modelagem Computacional, da Universidade Federal de Juiz de Fora como requisito parcial à obtenção do grau de Mestre em Modelagem Computacional.

Aprovada em 26 de junho de 2015.

BANCA EXAMINADORA

Prof. D.Sc. Bernardo Martins Rocha - Orientador
Universidade Federal de Juiz de Fora

Prof. D.Sc. Rodrigo Weber dos Santos - Coorientador
Universidade Federal de Juiz de Fora

Prof. D.Sc. Gilson Antônio Giraldi
Laboratório Nacional de Computação Científica

Prof. D.Sc. Marcelo Lobosco
Universidade Federal de Juiz de Fora

Prof. D.Sc. Rafael Alves Bonfim de Queiroz
Universidade Federal de Juiz de Fora

Dedico este trabalho aos meus pais.

AGRADECIMENTOS

Aos meus pais e à minha irmã pelo apoio, encorajamento e ajuda para que eu pudesse concluir este trabalho. À minha namorada, Lahis pela paciência e total apoio.

Aos amigos do Prédio Azul: Evelyn, Gustavo, Carla, Dani, Lemão, João, Johnny, Lucas, Sabrina, Ana Amélia, Anna Cláudia, Janaína, Zé que sempre estavam unidos ajudando uns aos outros e se divertindo também.

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

Ao professor Rodrigo que foi coorientador deste trabalho, cuja contribuição foi muito importante para realização do presente trabalho.

Aos professores do Programa de Pós Graduação em Modelagem Computacional pelos seus ensinamentos e aos funcionários do curso, que durante esses anos, contribuíram de algum modo para o nosso enriquecimento pessoal e profissional.

Agradeço também à CAPES pelo suporte financeiro.

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

Albert Einstein

RESUMO

Este trabalho apresenta o método de lattice Boltzmann (MLB) para simulações computacionais da atividade elétrica cardíaca usando o modelo monodomínio. Uma implementação otimizada do método de lattice Boltzmann é apresentada, a qual usa um modelo de colisão com múltiplos parâmetros de relaxação conhecido como *multiple relaxation time* (MRT), para considerar a anisotropia do tecido cardíaco. Com foco em simulações rápidas da dinâmica cardíaca, devido ao alto grau de paralelismo presente no MLB, uma implementação que executa em uma unidade de processamento gráfico (GPU) foi realizada e seu desempenho foi estudado através de domínios tridimensionais regulares e irregulares. Os resultados da implementação para simulações cardíacas mostraram fatores de aceleração tão altos quanto 500× para a simulação global e para o MLB um desempenho de 419 *mega lattice update per second* (MLUPS) foi alcançado. Com tempos de execução próximos ao tempo real em um único computador equipado com uma GPU moderna, estes resultados mostram que este trabalho é uma proposta promissora para aplicação em ambiente clínico.

Palavras-chave: Método de lattice Boltzmann. Eletrofisiologia cardíaca. Monodomínio. Computação de Alto Desempenho.

ABSTRACT

This work presents the lattice Boltzmann method (LBM) for computational simulations of the cardiac electrical activity using monodomain model. An optimized implementation of the lattice Boltzmann method is presented which uses a collision model with multiple relaxation parameters known as multiple relaxation time (MRT) in order to consider the anisotropy of the cardiac tissue. With focus on fast simulations of cardiac dynamics, due to the high level of parallelism present in the LBM, a GPU parallelization was performed and its performance was studied under regular and irregular three-dimensional domains. The results of our optimized LBM GPU implementation for cardiac simulations shown acceleration factors as high as 500× for the overall simulation and for the LBM a performance of 419 mega lattice updates per second (MLUPS) was achieved. With near real time simulations in a single computer equipped with a modern GPU these results show that the proposed framework is a promising approach for application in a clinical workflow.

Keywords: Lattice Boltzmann method. Cardiac electrophysiologoy. Monodomain. High performace Computation.

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

Lista de Abreviações

ATP	Trifosfato de adenosina
BGK	Bhatnagar, Gross, Krook
CPU	Unidade Central de Processamento
CUDA	Compute Unified Device Architecture
EDO	Equação Diferencial Ordinária
EDP	Equação Diferencial Parcial
GPU	Unidade de Processamento Gráfico
GPGPU	Unidade de Processamento Gráfico de propósito geral
LGAC	Lattice Gas Autômato Celular
LRI	Luo-Rudy
MDF	Método das Diferenças Finitas
MEF	Método dos Elementos Finitos
MLB	Método de Lattice Boltzmann
MLUPS	*Mega lattice updates per second*
MPI	Message Passing Interface
MS	Mitchell-Schaeffer
MSH	Mahajan-Shiferaw
MRT	*Multiple relaxation time*
MVF	Método dos Volumes Finitos
RS	Rush-Larsen
SDK	*Software Development Kit*
SM	*Streaming multiprocessor*
SP	Processador escalar
SRT	*Single relaxation time*
TT2	ten Tusscher-Panfilov

1 Introdução

O coração é responsável pelo bombeamento do sangue, transportando-o para o corpo. Uma falha neste processo de bombeamento pode levar à morte súbita, que é considerada uma das principais causas de morte no mundo. A morte súbita pode ser provocada por arritmias, que são alterações no sistema elétrico do coração produzindo ritmos anormais do batimento cardíaco. Este ritmo irregular pode fazer com que os impulsos elétricos acelerem a contração dos músculos cardíacos ou provocar um comportamento caótico na contração do tecido cardíaco.

A atividade elétrica está diretamente relacionada com a contração do coração e consequentemente relacionada com a função de bombeamento do sangue, portanto o estudo da eletrofisiologia cardíaca é fundamental para o entendimento do funcionamento do coração, assim como na busca de tratamentos para doenças cardíacas ou no auxílio em cirurgias.

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 destes modelos, 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 e Panfilov, 2004).

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

Existem diversos métodos numéricos que podem ser utilizados para a resolução destes modelos, alguns exemplos são o método das diferenças finitas (MDF) (LeVeque, 2007), método dos elementos finitos (MEF) (Hughes, 2000) e método dos volumes finitos (MVF) (Eymard *et al.*, 2000). Tais métodos se baseiam na discretização das equações que descrevem o fenômeno macroscopicamente e os sistemas resultantes são resolvidos usando métodos numéricos tradicionais tais como decomposição LU ou métodos iterativos como o método do Gradiente Conjugado (Trefethen e Bau, 1997).

Outra classe de métodos, que vem sendo bastante utilizados para simulação computacional de fluidos, envolve os métodos *Lattice-Gas* Autômato Celular (LGAC) (Wolf-Gladrow, 2000) e o Método de *lattice* Boltzmann (MLB) (Mohamad, 2011). Tais métodos partem das propriedades microscópicas para descrever o fenômeno na escala macroscópica. Apesar do 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 de reação-difusão como a eletrofisiologia cardíaca, que são fenômenos de grande interesse de pesquisa.

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 sua implementação pode ser paralelizada facilmente, permitindo obter um grande ganho em eficiência computacional. Neste trabalho o MLB aplicado à eletrofisiologia cardíaca será paralelizado utilizando a plataforma CUDA, que é executada em unidades de processamento gráfico.

O principal objetivo deste trabalho é estudar, analisar e implementar o MLB, para simulações de fenômenos físicos complexos como problemas de reação-difusão. O MLB é um método recente que vem se mostrando atrativo 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 implementação do método de *lattice* Boltzmann aplicado ao problema da eletrofisiologia cardíaca. Será feita a implementação do MLB e a validação da solução encontrada para o problema proposto através de testes com um *benchmark* de eletrofisiologia. Mostrando assim, que é possível utilizar o MLB para a solução de modelos matemáticos da eletrofisiologia cardíaca. Em seguida, devido ao alto custo computacional, procura-se otimizar a implementação, através do uso de computação paralela e também aplicando otimizações ao algoritmo do MLB. Então será avaliado o desempenho do método neste ambiente. Com a implementação otimizada serão realizados testes realísticos, simulando geometrias irregulares complexas, que reproduzem os ventrículos de um coelho, além de considerar a anisotropia do tecido cardíaco. Serão apresentados os bons desempenhos alcançados com esta implementação, com acelerações de até 500× e tempo de execução próximo ao tempo real.

O único trabalho encontrado que utiliza o MLB para a solução de problemas da eletrofisiologia cardíaca foi Rapaka *et al.* (2012), que realizou uma implementação sequencial do MLB usando o modelo celular de Mitchell e Schaeffer (2003). Portanto, as contribuições do presente trabalho estão na implementação paralela para geometrias tridimensionais complexas, com tempos de execução próximos ao tempo real. Além disso, o MLB foi validado para problemas de eletrofisiologia cardíaca através de outra metodologia e foram utilizados modelos celulares mais complexos. Outra contribuição é utilização do Algoritmo *swap* para a etapa de propagação, que economiza memória e é

mais rápido do que o algoritmo tradicional. Em relação ao desempenho, este trabalho obteve alguns resultados melhores do que outros trabalhos, que podem ser vistos com mais detalhes na Seção 7.1. E parte deste trabalho foi publicada no artigo Campos *et al.* (2015).

1.1 Organização da dissertação

O segundo capítulo trata da modelagem da eletrofisiologia cardíaca, apresentando os modelos matemáticos que serão utilizados. O terceiro capítulo aborda os aspectos teóricos do método de *lattice* Boltzmann, apresentando a derivação da equação que descreve a dinâmica do método, além de apresentar a equação do método para a solução de problemas de escoamentos de fluidos e também a equação para a solução de problemas de reação-difusão anisotrópica. O quarto capítulo apresenta de forma sucinta as técnicas de computação paralela utilizadas neste trabalho e as métricas para avaliar o desempenho da implementação paralela. O quinto capítulo apresenta os detalhes sobre a implementação do método de *lattice* Boltzmann para a simulação de um problema da eletrofisiologia cardíaca. Abordando detalhes sobre as estruturas de dados e otimizações realizadas. O sexto capítulo apresenta os resultados encontrados para os cinco experimentos realizados, validando o MLB para a simulação de problemas da eletrofisiologia cardíaca e avaliando o desempenho da implementação paralela em diferentes cenários. O último capítulo apresenta as conclusões alcançadas com este trabalho, compara os resultados obtidos com trabalhos relacionados e descreve algumas ideias de trabalhos futuros. O Apêndice A apresenta resultados complementares para o experimento do *benchmark.*

2 Modelagem da eletrofisiologia cardíaca

As células do músculo cardíaco pertencem a uma classe de células conhecidas como células excitáveis, que tem a habilidade de responder ativamente a estímulos elétricos. As células do coração são conectadas de maneira que uma célula estimulada pode passar um sinal elétrico para as células vizinhas. Esta habilidade permite que a estimulação de uma parte do coração se propague por todo o músculo cardíaco.

A propagação do sinal em tecidos excitáveis leva à chamada despolarização das células. Quando as células estão em repouso, existe uma diferença de potencial através da membrana celular. Quando células excitáveis são estimuladas eletricamente, elas despolarizam, ou seja, o potencial intracelular sai de um valor negativo e passa a ser positivo ou próximo de zero. A despolarização é um processo muito rápido e é seguida de um processo lento, a repolarização, que restaura a diferença de potencial para o seu valor de repouso. O ciclo completo de despolarização e repolarização, chamado de potencial de ação, pode ser visto na Figura 2.1. É ele que inicia a contração das células musculares e que permite as células do sistema nervoso ou células cardíacas responderem a um sinal elétrico de entrada, passando o sinal às células vizinhas. Esta diferença de potencial

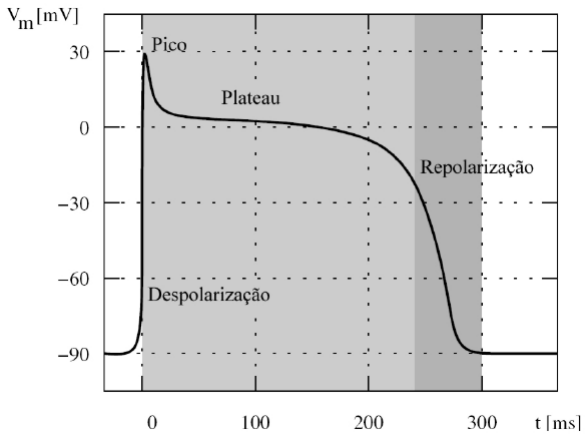


Figura 2.1: Ilustração de um potencial de ação (extraído de Oliveira (2013)).

através da membrana celular é essencial para o comportamento de tecidos excitáveis, e para compreender o funcionamento do coração é preciso construir modelos matemáticos

que sejam capazes de descrever esta diferença de potencial.

A atividade elétrica do coração está diretamente ligada à função do órgão, que é de bombear o sangue para o restante do corpo, levando nutrientes e oxigênio às células. O bombeamento do sangue é realizado pela contração do coração, que acontece devido ao potencial de ação. Este é gerado no átrio direito, como mostrado na Figura 2.2, em uma região denominada nodo sinoatrial, que é responsável pelo ritmo do coração. Para que o sangue seja bombeado de forma correta, todos os miócitos devem se contrair ao mesmo tempo e para que isso ocorra, o potencial de ação deve ser propagado pelo tecido cardíaco. Após sua geração no nodo sinoatrial, ele chega aos átrios direito, esquerdo e depois para os ventrículos, através das fibras de Purkinje e do septo interventricular. Quando o coração está em condições normais este processo se repete várias vezes, levando a batimentos em uma certa frequência, porém este processo pode ser alterado por doenças cardíacas, provocando arritmias que podem levar a morte.

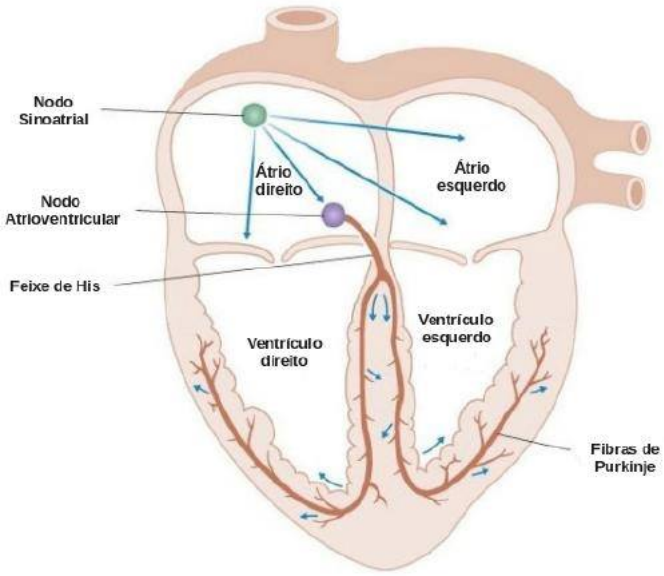


Figura 2.2: Propagação do potencial de ação pelo tecido do coração (extraído de Constanzo (2007)).

2.1 Fisiologia da membrana celular

A membrana celular separa o meio extracelular do intracelular e consiste de uma bicamada fosfolipídica, onde uma extremidade tem afinidade com água (hidrofílica) e a outra não possui afinidade (hidrofóbica). A parte hidrofílica da membrana fica para fora e a parte hidrofóbica para dentro, deixando a membrana impermeável. Apesar da membrana ser

impermeável, existem proteínas capazes de formar canais que podem transportar íons de um meio para outro, como mostrado na Figura 2.3. Algumas proteínas de transporte

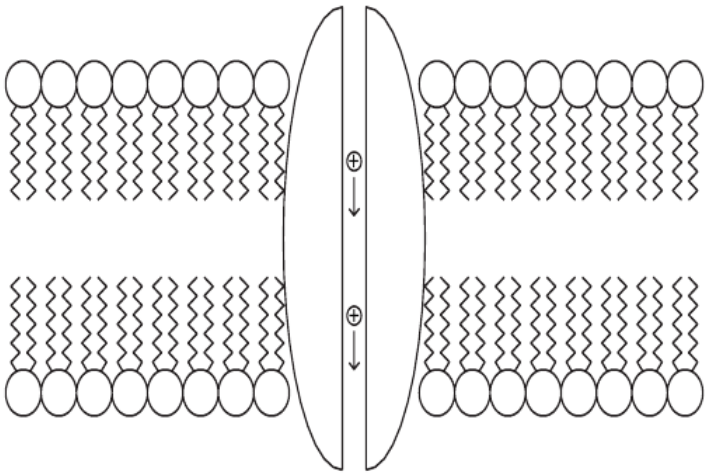


Figura 2.3: Membrana celular com proteína formando um canal para passagem de íons (extraído de Sundnes (2006)).

formam trocadores e bombas, que são importantes para a manutenção da concentração iônica correta nas células. Bombas e trocadores têm a capacidade de transportar íons na direção oposta ao fluxo gerado pelo gradiente de concentração e campo elétrico. Este transporte contra o fluxo pode ser feito usando o gradiente de concentração de um íon diferente ou pelo consumo de energia armazenada quimicamente na forma de ATP.

Além disso, certas proteínas de transporte formam canais iônicos na membrana por onde íons podem passar. O fluxo de íons através destes canais é passivo, sendo dirigido por gradientes de concentração e campos elétricos. Os canais são altamente seletivos à passagem de íons, permitindo que apenas um íon ou um pequeno grupo passe por um certo canal. E eles também têm a habilidade de abrir e fechar em resposta à troca na direção do campo elétrico e concentração iônica (Sundnes, 2006).

Esse transporte de substâncias através da membrana gera diferenças nas concentrações elétrica e química nos meios intra e extracelular, consequentemente gerando uma diferença de potencial na membrana, denominado potencial transmembrânico, que tem papel fundamental na comunicação intercelular (Oliveira, 2013).

2.2 Modelagem da membrana celular

O comportamento elétrico da membrana celular pode ser representado por um circuito resistor-capacitor, com o resistor não linear R_m reproduzindo a resistência elétrica da membrana, que ocorre devido a dinâmica dos canais iônicos, e um capacitor C_m que representa a separação de cargas entre os meios intra e extracelular realizada pela membrana. A Figura 2.4 apresenta o circuito descrito, onde pode-se observar o fluxo de duas correntes em paralelo: a corrente capacitiva I_c e a corrente iônica I_{ion}, que representa a soma das correntes referentes a cada canal iônico considerado no modelo. O potencial sobre a membrana v é dependente da carga Q:

$$v = \frac{Q}{C_m} \tag{2.1}$$

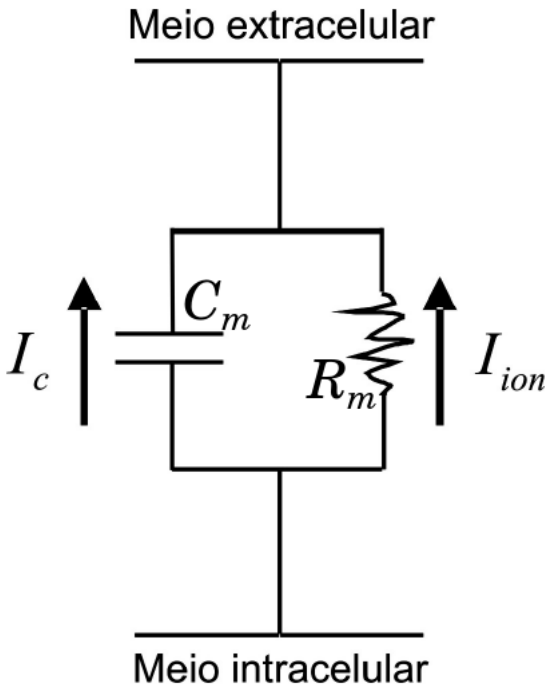


Figura 2.4: Modelagem da membrana através de um circuito resistor-capacitor (extraído de Campos (2008)).

Uma variação no potencial transmembrânico v pode ser descrito por um fluxo de corrente I_c:

$$\frac{dv}{dt} = \frac{d}{dt}\frac{Q}{C_m} = \frac{I_c}{C_m} \tag{2.2}$$

assumindo que a capacitância C_m é contante no tempo. A corrente transmembrânica total

I_m pode ser calculada, tomando a soma das correntes iônica e capacitiva:

$$I_m = I_{ion} + I_c \tag{2.3}$$

Usando (2.2), temos:

$$I_m = I_{ion} + C_m \frac{dv}{dt} \tag{2.4}$$

Considerando que a célula está isolada, não há acúmulo de carga em nenhum dos lados da membrana, portanto $I_m = 0$ e chegamos na seguinte equação:

$$C_m \frac{dv}{dt} + I_{ion} = 0 \tag{2.5}$$

Para completar o modelo celular é necessário definir a corrente iônica, que descreve a dinâmica dos canais iônicos considerados. Estes podem ser modelados com estados e funções que descrevem a transição entre eles, normalmente resultando em um sistema de equações diferenciais ordinárias. Então o modelo celular é descrito pelo seguinte sistema:

$$C_m \frac{dv}{dt} = -I_{ion}(t, v, \boldsymbol{\eta}), \tag{2.6}$$

$$\frac{d\eta}{dt} = f(v, \boldsymbol{\eta}), \tag{2.7}$$

onde $\boldsymbol{\eta}$ é um vetor das variáveis de estado, que representa o estado de cada canal iônico considerado no modelo, que podem ser ativados ou desativados dependendo do potencial, do tempo ou ainda da concentração dos íons. Os modelos celulares utilizados neste trabalho são apresentados na Seção 2.3.3. Maiores informações sobre modelagem eletrofisiológica podem ser encontrada em Keener e Sneyd (1998); Sachse (2004); Sundnes (2006).

2.3 Propagação elétrica no tecido

O corpo humano consiste de bilhões de células, que podem estar conectadas por vários mecanismos de acoplamento, dependendo do tipo de tecido. Quando construímos modelos matemáticos para a atividade elétrica no tecido, uma abordagem é modelar cada célula como uma unidade separada e acoplá-las usando modelos matemáticos para os mecanismos de acoplamento. O tecido cardíaco é composto de miócitos interconectados, acoplados

através de um meio condutivo intracelular, que é cercado por um fluido condutivo no espaço extracelular entre as células. Matematicamente este fenômeno pode ser modelado como dois espaços interpenetrantes (espaços intra e extracelular) que ocupam o mesmo volume e são separados pela membrana celular.

O modelo matemático mais completo da atividade elétrica cardíaca é o modelo bidomínio que considera os meios intra e extracelular como quantidades médias acopladas pela corrente transmembrânica. Este modelo pode ser formalmente derivado considerando a conservação das correntes intra e extracelular, pelo acoplamento dos dois domínios com a corrente transmembrânica e fazendo uso das leis de Ohm e Kirchhoff, que é detalhado em Keener e Sneyd (1998).

2.3.1 Modelo do Bidomínio

O modelo do bidomínio descreve o tecido cardíaco de forma homogeneizada (Keener e Sneyd, 1998), considerando que o tecido é dividido em dois domínios separados: o intracelular e o extracelular, que são considerados contínuos. A justificativa para que o domínio intracelular seja contínuo é que as células são conectadas por junções do tipo *gap*. Estas junções são pequenos canais embutidos na membrana que formam contato direto entre as células vizinhas.

Em cada domínio é definido um potencial elétrico, que em cada ponto deve ser visto como uma quantidade média sobre um pequeno volume. Como consequência dessa definição, temos que cada ponto do tecido cardíaco se encontra em ambos os domínios e portanto cada ponto possui um potencial intracelular v_i e um potencial extracelular v_e. Suas correntes são dadas por:

$$J_i = -\boldsymbol{\sigma_i} \nabla v_i \tag{2.8}$$

$$J_e = -\boldsymbol{\sigma_e} \nabla v_e \tag{2.9}$$

onde $\boldsymbol{\sigma_i}$ e $\boldsymbol{\sigma_e}$ são as condutividades nos respectivos meios. Assumindo que não existe acúmulo de carga em qualquer ponto, temos que a corrente total se conserva

$$\nabla \cdot J_t = \nabla \cdot J_i + \nabla \cdot J_e = 0, \tag{2.10}$$

que pode ser escrito em função dos potenciais como

$$\nabla \cdot (\boldsymbol{\sigma_i} \nabla v_i) + \nabla \cdot (\boldsymbol{\sigma_e} \nabla v_e) = 0, \tag{2.11}$$

Lembrando da Equação (2.4), temos que a corrente transmembrânica é a soma de uma corrente capacitiva e uma corrente iônica. Além disso, pode-se observar que toda corrente que deixa o meio intracelular é uma corrente transmembrânica, portanto temos:

$$\nabla \cdot (\boldsymbol{\sigma_i} \nabla v_i) = \chi \left(I_{ion} + C_m \frac{dv}{dt} \right). \tag{2.12}$$

A corrente iônica é convenientemente medida por unidade de área da membrana celular, enquanto densidade de carga e corrente são medidas por unidade de volume. A constante χ representa a área da membrana celular por unidade de volume.

O potencial transmembrânico é definido como a diferença entre os potenciais intracelular e extracelular $v = v_i - v_e$. Pode-se usar esta relação para rearranjar as equações do bidomínio, de maneira que o modelo fique em função apenas de v_e e v:

$$\nabla \cdot (\boldsymbol{\sigma_i} \nabla v) + \nabla \cdot (\boldsymbol{\sigma_i} \nabla v_e) = \chi \left(I_{ion} + C_m \frac{dv}{dt} \right), \tag{2.13}$$

$$\nabla \cdot (\boldsymbol{\sigma_i} \nabla v) + \nabla \cdot ((\boldsymbol{\sigma_i} + \boldsymbol{\sigma_e}) \nabla v_e) = 0. \tag{2.14}$$

As equações (2.13) e (2.14) formam a formulação padrão do bidomínio, que devem possuir condições iniciais e de contorno apropriadas (Sundnes, 2006).

2.3.2 Modelo Monodomínio

Como o modelo bidomínio consiste de um sistema complexo de EDPs e possui solução numérica custosa, é comum assumir que a taxa de anisotropia no domínio extracelular é proporcional à taxa de anisotropia do domínio intracelular para obter um modelo simplificado, chamado modelo monodomínio. Seguindo esta consideração, o modelo monodomínio pode ser obtido por uma redução do modelo bidomínio e é inteiramente escrito em termos do potencial transmembrânico.

Considere que as taxas de anisotropia nos dois domínios são iguais, ou seja, $\boldsymbol{\sigma_e} = \lambda \boldsymbol{\sigma_i}$, onde λ é uma constante. Então, $\boldsymbol{\sigma_e}$ pode ser eliminado das equações do modelo, resultando

em

$$\nabla \cdot (\boldsymbol{\sigma_i} \nabla v) + \nabla \cdot (\boldsymbol{\sigma_i} \nabla v_e) = \chi \left(I_{ion} + C_m \frac{dv}{dt} \right), \tag{2.15}$$

$$\nabla \cdot (\boldsymbol{\sigma_i} \nabla v) + (1 + \lambda) \nabla \cdot (\boldsymbol{\sigma_i} \nabla v_e) = 0. \tag{2.16}$$

Da Equação 2.16, chega-se em

$$\nabla \cdot (\boldsymbol{\sigma_i} \nabla v_e) = -\frac{1}{1 + \lambda} \nabla \cdot (\boldsymbol{\sigma_i} \nabla v), \tag{2.17}$$

e substituindo esta equação em 2.15 e rearranjando os termos chega-se à formulação padrão do modelo monodomínio:

$$\frac{\lambda}{1 + \lambda} \nabla \cdot (\boldsymbol{\sigma_i} \nabla v) = \chi \left(I_{ion} + C_m \frac{dv}{dt} \right). \tag{2.18}$$

O monodomínio possui vantagens em relação ao custo da solução numérica, mas possui limitações como a hipótese de anisotropias iguais. Além disso, é difícil especificar o valor de λ de maneira a reproduzir o comportamento fisiológico.

O problema completo do monodomínio é dado por:

$$\begin{cases} \chi \left(C_m \dfrac{\partial v}{\partial t} + I_{ion}(v, \boldsymbol{\eta}) \right) = \nabla \cdot (\boldsymbol{\sigma} \nabla v), \\ \mathbf{n} \cdot \boldsymbol{\sigma} \nabla v = 0, \\ v(\mathbf{x}, 0) = v_0(\mathbf{x}), \quad \boldsymbol{\eta}(\mathbf{x}, 0) = \boldsymbol{\eta}_0(\mathbf{x}), \end{cases} \tag{2.19}$$

onde v é a variável de interesse e representa o potencial transmembrânico, χ 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 $\boldsymbol{\eta}$, que é descrito por um sistema de EDOs para controlar a cinética das variáveis de estado. O primeiro termo do lado esquerdo é a taxa de mudança do potencial transmembrânico no tempo, enquanto o termo do lado direito descreve a difusão. As condições iniciais para v e $\boldsymbol{\eta}$ são dadas por $v_0(\mathbf{x})$ e $\boldsymbol{\eta}_0(\mathbf{x})$, respectivamente. 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, como mostra a segunda equação do problema (2.19), onde $\mathbf{n}$ é o vetor normal ao contorno.

Estudos anatômicos mostraram que o músculo do coração é um material fortemente anisotrópico, devido ao fato do músculo cardíaco ser constituído de fibras, como mostrado na Figura 2.5. As fibras são organizadas em folhas, o que leva a três direções características para a condutividade no tecido cardíaco: paralela às fibras; perpendicular às fibras, mas paralela às folhas; e perpendicular às folhas. Assim, a propagação elétrica é mais rápida na direção da fibra do que nas direções transversais à fibra.

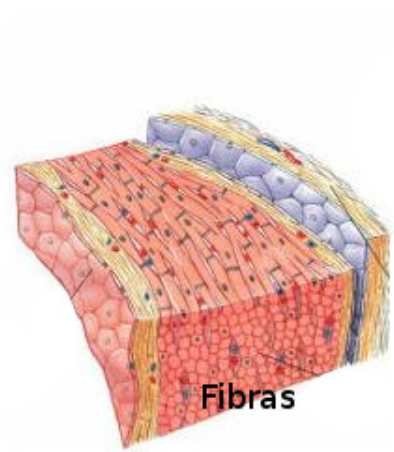


Figura 2.5: Organização das fibras do tecido cardíaco. (Adaptado de SoBiologia (2015))

No modelo do monodomínio, $\boldsymbol{\sigma}$ é o tensor de condutividade que descreve as propriedades elétricas do tecido. Se for considerado que o tecido é um material transversalmente isotrópico, ou seja, a direção da fibra $\mathbf{a}_l$ é a de maior condutividade, então a condutividade é dada por

$$\boldsymbol{\sigma}(\mathbf{x}) = \sigma_t \mathbf{I} + (\sigma_l - \sigma_t)\mathbf{a}_l(\mathbf{x})\mathbf{a}_l^T(\mathbf{x}), \tag{2.20}$$

onde $\mathbf{I}$ é a matriz identidade de dimensão 3×3, σ_l e σ_t são os valores da condutividade na direção da fibra e na direção transversal à fibra, respectivamente.

2.3.3 Modelos celulares

Para completar a descrição da equação (2.19) 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 e Huxley (1952).

Nesse trabalho iremos apresentar simulações da atividade elétrica cardíaca utilizando quatro modelos celulares diferentes, com níveis de complexidade distintos. 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 e Schaeffer, 2003). Por outro lado, o modelo de Luo-Rudy (LRI) reproduz o potencial de ação de células ventriculares do preá-da-índia (Luo, C. e Rudy, Y., 1991). Outro modelo utilizado foi o de ten Tusscher-Panfilov (TT2) (ten Tusscher e Panfilov, 2006), que descreve o potencial de ação de células do ventrículo humano. Por último, foi utilizado o modelo Mahajan *et al.* (2008) (MHS), que apresenta a geração do potencial de ação para células ventriculares de coelhos.

2.3.4 Modelo de Mitchell-Schaeffer

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

$$I_{ion} = I_{in}(v,h) + I_{out}(v), \tag{2.21}$$

$$\frac{dv}{dt} = I_{ion} + I_{stim}(t), \tag{2.22}$$

$$\frac{dh}{dt} = \begin{cases} \frac{1-h}{\tau_{open}} & \text{se } v < v_{gate} \\ \frac{-h}{\tau_{close}} & \text{se } v > v_{gate} \end{cases} \tag{2.23}$$

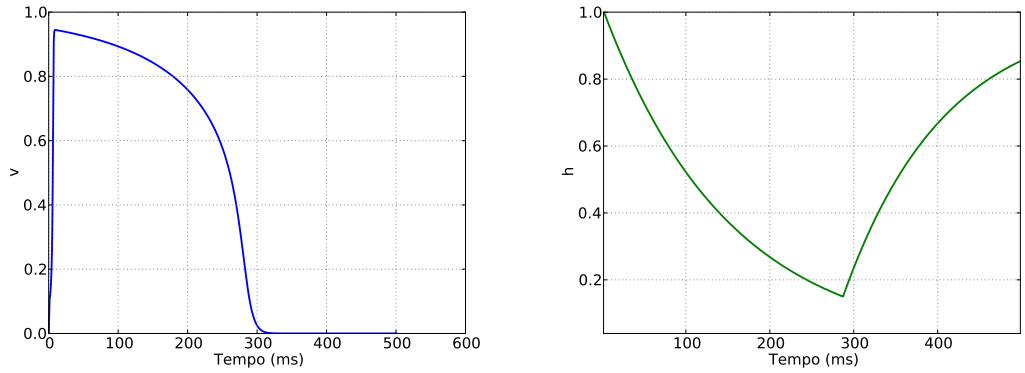
onde τ_{in}, τ_{out}, τ_{open}, τ_{close} e v_{gate} são constantes, e as correntes I_{in} e I_{out} são dadas por

$$I_{in}(v,h) = \frac{hC(v)}{\tau_{in}} \tag{2.24}$$

$$I_{out}(v) = -\frac{v}{\tau_{out}} \tag{2.25}$$

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 2.6 apresenta o gráfico das variáveis v e h ao longo do tempo.

Veja mais detalhes sobre o modelo em Mitchell e Schaeffer (2003).



(a) (b)

Figura 2.6: Exemplo de simulação do modelo de Mitchell-Schaeffer durante 500 ms, com um estímulo inicial aplicado de $I_{stim} = 0.1$, usando o método de Euler. (a) Potencial transmembrânico (variável v) e (b) variável de estado h.

2.3.5 Modelo de Luo-Rudy

Luo-Rudy (LRI) é um modelo detalhado que representa de forma quantitativa a geração do potencial de ação de células ventriculares do preá-da-índia, incluindo a atividade de diferentes correntes iônicas, como mostrado na Figura 2.7. No modelo LRI, o termo I_{ion} é definido como:

$$I_{ion}(v, \boldsymbol{\eta}) = I_{Na} + I_{si} + I_K + I_{K1} + I_{Kp} + I_b, \tag{2.26}$$

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.

As correntes I_{Na}, I_{si}, I_K e I_{K1} na Eq. (2.26) são controladas por variáveis descritas por EDOs que são da forma:

$$\frac{dn}{dt} = \frac{n_\infty - n}{\tau_n}, \tag{2.27}$$

onde os termos n_∞ e τ_n são definidos como

$$n_\infty = \frac{\alpha}{\alpha + \beta}, \quad \tau_n = \frac{1}{\alpha + \beta}, \tag{2.28}$$

sendo que α e β são funções de v. A Figura 2.8(a) apresenta o potencial de ação, enquanto a Figura 2.8(b) apresenta algumas das correntes iônicas descritas na Eq. (2.26) do modelo

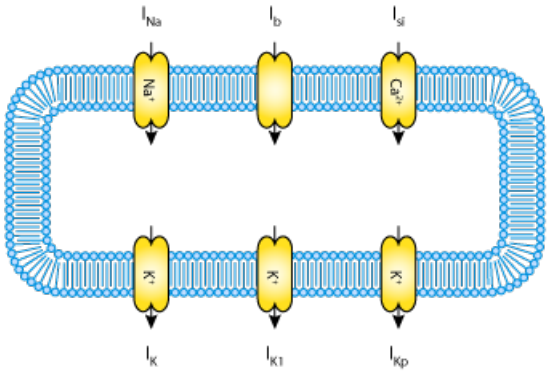
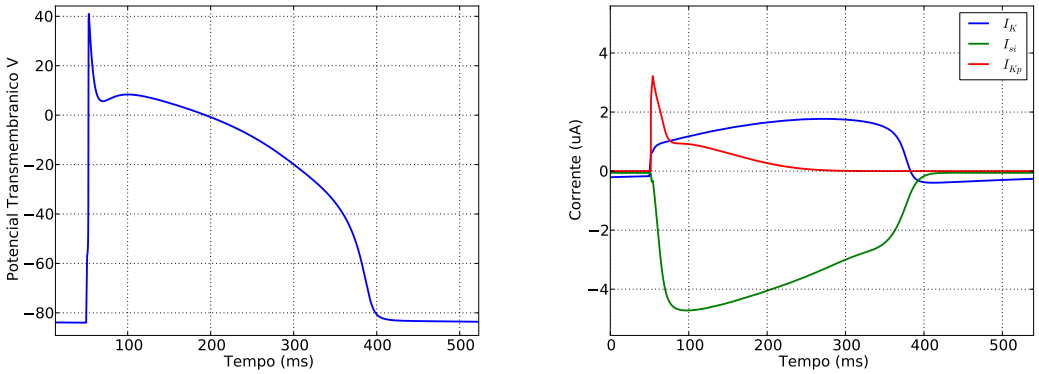


Figura 2.7: Esquema do modelo celular LRI. (extraído de Lloyd *et al.* (2008))

LRI. A descrição completa do sistema de EDOs, suas variáveis e parâmetros pode ser encontrada em Luo, C. e Rudy, Y. (1991).



(a) (b)

Figura 2.8: Exemplo de simulação do modelo de Luo-Rudy durante 500 ms, inicializado com um estímulo inicial aplicado em $t = 50$ ms. (a) Potencial transmembrânico v e (b) algumas correntes iônicas (I_K, I_{si} e I_{Kp}) do modelo de Luo-Rudy.

2.3.6 Modelo ten Tusscher-Panfilov

Um dos modelos matemáticos mais usados para simulações computacionais de células ventriculares humanas é o modelo de ten Tusscher e Panfilov (2006) (TT2). Este modelo tem 19 variáveis de estado que são descritas por EDOs, descrevendo também a concentração de cálcio e condutividades dos canais iônicos, como apresenta a Figura 2.9.

A corrente iônica total I_{ion} deste modelo é dada pela soma das seguintes correntes:

$$I_{ion} = I_{Na} + I_{K1} + I_{to} + I_{Kr} + I_{Ks} + I_{CaL} + I_{NaCa} + I_{NaK} + I_{pCa} + I_{pK} + I_{Ca,b} + I_{Na,b}. \quad (2.29)$$

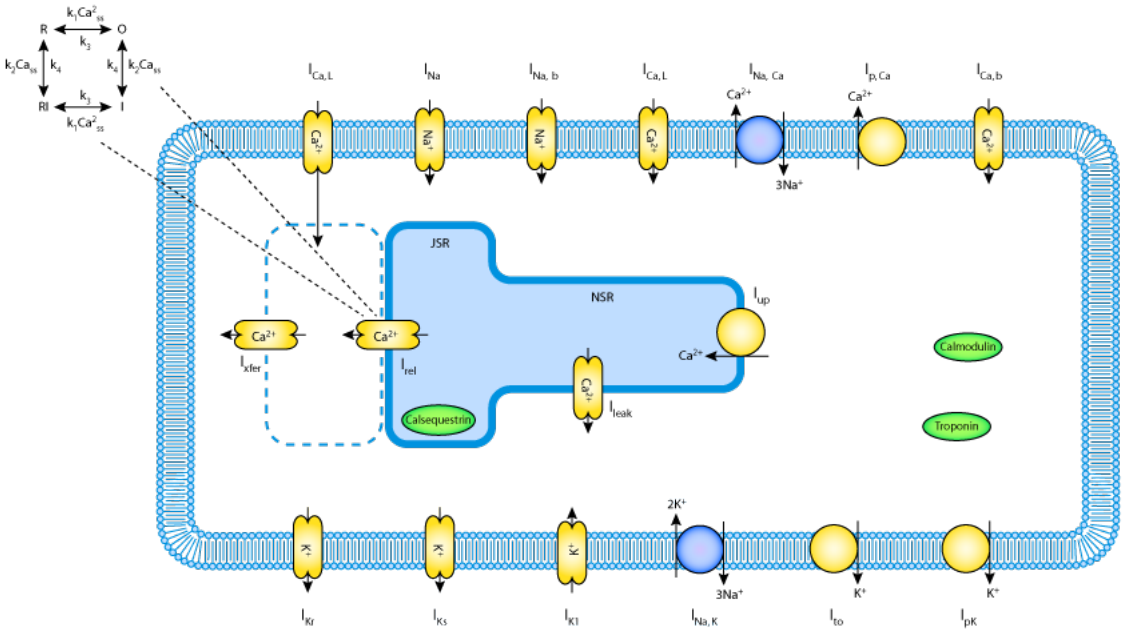


Figura 2.9: Representação do modelo celular TT2. (extraído de Lloyd *et al.* (2008))

Uma descrição completa das correntes que formam a corrente iônica total, as equações do modelo e seus parâmetros podem ser encontrados na publicação original (ten Tusscher e Panfilov, 2006).

2.3.7 Modelo Mahajan-Shiferaw

O modelo Mahajan-Shiferaw (MSH) é uma modificação de um modelo ventricular de coelho anterior para reproduzir o potencial de ação em taxas rápidas, relevantes para o estudo de taquiarritmias ventriculares. O modelo consiste de 26 EDOs não lineares, incluindo tensão, variáveis de gatilho, estados de Markov para corrente de cálcio tipo L, *buffers* de troponina, concentração média de cálcio e outras.

A corrente iônica total I_{ion} deste modelo é dada pelo somatório das correntes:

$$I_{ion} = I_{Na} + I_{to,f} + I_{to,s} + I_{Kr} + I_{Ks} + I_{K1} + I_{NaK} + I_{Ca} + I_{NaCa}. \quad (2.30)$$

O potencial de ação reproduzido pelo modelo MSH é apresentado na Figura 2.10. Para uma descrição completa do modelo veja Mahajan *et al.* (2008).

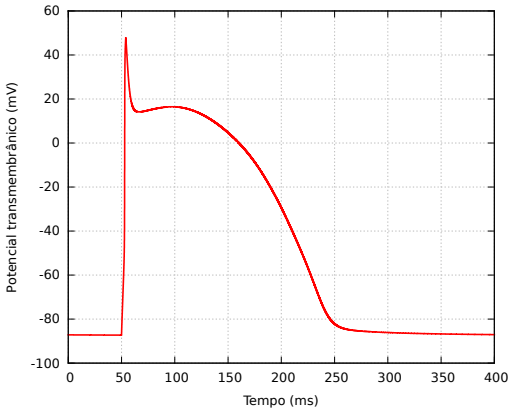


Figura 2.10: Potencial de ação do modelo MSH.

2.4 Solução numérica dos modelos celulares

Os modelos celulares apresentados são sistemas de EDOs, que não possuem solução analítica ou são de difícil solução, portanto é necessário usar algum método numérico para encontrar a solução deste sistema. Nesta seção são apresentados dois métodos numéricos que podem ser usados na resolução destas EDOs, sendo que o método de Rush-Larsen foi utilizado neste trabalho.

2.4.1 Euler

O método de Euler é usado para encontrar a solução de problemas de valor inicial, que possuem a forma

$$u' = f(u,t), \tag{2.31}$$

$$u(0) = u_0. \tag{2.32}$$

O método pode ser obtido escrevendo a solução do problema $u(t)$ usando série de Taylor em torno do ponto t_n

$$u(t) = u(t_n) + u'(t_n)(t-t_n) + u''(t_n)\frac{(t-t_n)^2}{2!} + \cdots \tag{2.33}$$

Avaliando em t_{n+1} resulta-se em

$$u(t_{n+1}) = u(t_n) + u'(t_n)(t_{n+1} - t_n) + u''(t_n)\frac{(t_{n+1} - t_n)^2}{2!} + \cdots \tag{2.34}$$

truncando a série no segundo termo e considerando $(t_{n+1} - t_n) = \Delta t$ e $u' = f(u(t), t)$ chega-se em

$$u(t_{n+1}) \approx u(t_n) + \Delta t f(u(t_n), t_n), \tag{2.35}$$

de onde se obtém o método de Euler explícito, com convergência linear

$$u_{n+1} = u_n + \Delta t f(u_n, t_n). \tag{2.36}$$

O método de Euler apresentado é explícito, pois depende apenas do valor de u no passo anterior para calcular u no passo atual. Em certos problemas, o método de Euler necessita de um passo de tempo Δt muito pequeno para que não ocorra instabilidade numérica. Nestes casos é mais eficiente utilizar métodos implícitos ou métodos explícitos com melhores propriedades de estabilidade.

2.4.2 Rush Larsen

A solução numérica do sistema de EDOs dado pelo modelo celular é realizado usando o método de Rush e Larsen (1978). RL é um método explícito que tira vantagem da estrutura das EDOs que tem a forma da equação (2.27) para a integração numérica. Além disso, ele tem melhores propriedades de estabilidade do que o método de Euler explícito, permitindo assim usar passos de tempo maiores, resultando em um método eficiente para a solução numérica de modelos celulares.

Dada uma equação da forma

$$\frac{dy^j}{dt} = \alpha_j(1 - y^j) - \beta_j y^j, \tag{2.37}$$

o método RL assume que α_j e β_j são aproximadamente constantes em um pequeno intervalo de tempo. Então é aplicada uma linearização local nestas equações e a solução

exata da EDO é usada para atualizar y_n^j.

$$y_{n+1}^j = \left(y_n^j - \frac{\alpha_j}{\alpha_j + \beta_j} \right) e^{-(\alpha_j + \beta_j)\Delta t} + \frac{\alpha_j}{\alpha_j + \beta_j} \tag{2.38}$$

Quando uma equação não tem a forma da equação (2.37), utiliza-se o método de Euler explícito para resolver a equação, o que faz com que o método RL seja de primeira ordem, mas com uma estabilidade maior do que o método de Euler.

3 Método de Lattice Boltzmann

Neste capítulo será apresentado o método de *lattice* Boltzmann, inicialmente, em sua forma mais utilizada, isto é, voltado para simulação de escoamento de fluidos e, em seguida, será apresentado o método adaptado para a simulação de equações de reação-difusão, como a equação que descreve a atividade elétrica cardíaca. Além disso, mostra-se a forma adotada para se considerar a anisotropia do tecido cardíaco e como impor as condições de contorno no método de *lattice* Boltzmann.

O método de *lattice* Boltzmann (MLB) é um método numérico baseado em equações cinéticas, que simulam a dinâmica dos fluidos e outros problemas complexos na escala mesoscópica, entre as escalas micro e macroscópica, afim de recuperar o seu comportamento macroscópico. Ele vem se tornando bastante popular para a simulação de escoamentos de fluidos, como descrito em Zou e He (1997), Golbert (2009) e Mohamad (2011).

Este capítulo começa com uma introdução histórica do precursor do MLB, chamado de *lattice gas* autômato celular (LGAC) que é usado para resolver problemas da dinâmica dos fluidos. Chega-se então, na equação do MLB partindo da equação do LGAC. Em seguida é apresentada uma outra maneira de deduzir o método de *lattice* Boltzmann para problemas de escoamentos de fluidos, a partir da equação de transporte de Boltzmann. E então apresenta-se a estratégia usada para resolver problemas de reação-difusão usando o MLB.

3.1 Lattice gas autômato celular

O método *Lattice Gas* Autômato Celular (LGAC) é um autômato celular (Wolf-Gladrow, 2000) 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 (Giraldi *et al.*, 2005). As velocidades dessas partículas são limitadas a um conjunto finito de direções, o qual depende do tipo de *lattice*.

Em cada ponto do *lattice* podem 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 a partir de uma regra (devido a uma colisão frontal de duas partículas, por exemplo). Na etapa de propagação, as partículas se deslocam para células vizinhas, 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. As variáveis assumem os valores 0 ou 1, 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:

$$n_i(\mathbf{x} + \mathbf{e}_i, t+1) = n_i(\mathbf{x}, t) + \Delta_i(n_i(\mathbf{x}, t)), \;\; i = 0, ..., l-1 \;\; , \tag{3.1}$$

onde Δ é o operador de colisão (Golbert, 2009). De acordo com esta 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.

3.2 Do LGAC para o MLB

Enquanto no LGAC os estados são discretos, no MLB que será apresentado a seguir, os estados são distribuições de probabilidade para a velocidade de uma partícula, portanto o método de *lattice* Boltzmann se encontra na mesoescala. Para migrar 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, pode-se 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 (3.1), chega-se na seguinte equação:

$$N_i(\mathbf{x} + \mathbf{e}_i \Delta x, t + \Delta t) = N_i(\mathbf{x}, t) + \langle \Delta_i(\mathbf{n}) \rangle, \tag{3.2}$$

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 em

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

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

$$f_i(\mathbf{x} + \mathbf{e}_i \Delta x, t + \Delta t) = f_i(\mathbf{x}, t) + \Omega_i(\mathbf{f}), \tag{3.4}$$

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 deduzida partindo-se da equação de Boltzmann, como será mostrado na próxima seção.

3.3 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 nas equações abaixo:

$$\mathbf{F} = m\frac{d\mathbf{v}}{dt}, \;\; \mathbf{v} = \frac{d\mathbf{x}}{dt}, \tag{3.5}$$

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

Entretanto, quando se consideram 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 de 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:

$$\frac{\partial f}{\partial t} + \mathbf{v} \cdot \nabla f + \frac{\mathbf{F}}{m} \frac{\partial f}{\partial \mathbf{v}} = Q(f), \tag{3.6}$$

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

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

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 $\mathbf{u}$, que denotam a densidade e a velocidade do fluido, são calculadas respectivamente, a partir dos momentos da distribuição f, e são dadas pelas equações:

$$\rho = \int f d\mathbf{v}, \quad \rho\mathbf{u} = \int \mathbf{v} f d\mathbf{v}. \tag{3.8}$$

Para se chegar à equação do método, primeiramente o espaço de velocidades é discretizado em um conjunto de velocidades $\mathbf{e}_i$, com $i = 0, 1, 2, 3, \ldots, l-1$, onde l é a dimensão do espaço de velocidade. Assim a equação de Boltzmann fica da seguinte forma:

$$\frac{\partial f_i}{\partial t} + \mathbf{e}_i \cdot \nabla f_i = Q_i(f), \quad i = 0, \ldots, l-1. \tag{3.9}$$

Pode-se aproximar as derivadas no tempo e no espaço da equação anterior utilizando diferença finitas (He e Luo, 1997) e, assim, chega-se em

$$f_i(\mathbf{x} + \mathbf{e}_i \Delta x, t + \Delta t) - f_i(\mathbf{x}, t) = Q_i(f), \tag{3.10}$$

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

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 (Bhatnagar, Gross, Krook) para este termo (Bhatnagar *et al.*, 1954).

3.3.1 Aproximação BGK

A aproximação BGK se baseia no princípio de que cada colisão modifica a função de distribuição das partículas $f(\mathbf{x}, \mathbf{v}, t)$ 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:

$$Q^{BGK}(f) = \frac{1}{\tau}(f^{eq}(\mathbf{x}, \mathbf{v}, t) - f(\mathbf{x}, \mathbf{v}, t)), \tag{3.11}$$

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

$$f^{eq} = \frac{\rho}{(\frac{2\pi}{3})^{\frac{D}{2}}} \exp\left[-\frac{3}{2}(\mathbf{v} - \mathbf{u}) \cdot (\mathbf{v} - \mathbf{u})\right], \tag{3.12}$$

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:

$$f^{eq} = \frac{\rho}{(\frac{2\pi}{3})^{\frac{D}{2}}} \exp\left[-\frac{3}{2}(\mathbf{v} \cdot \mathbf{v})\right] \left[1 + 3(\mathbf{v} \cdot \mathbf{u}) + \frac{9}{2}(\mathbf{v} \cdot \mathbf{u})^2 - \frac{3}{2}(\mathbf{u} \cdot \mathbf{u})\right] \tag{3.13}$$

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

$$f_i^{eq} = w_i \rho \left[1 + 3(\mathbf{e}_i \cdot \mathbf{u}) + \frac{9}{2}(\mathbf{e}_i \cdot \mathbf{u})^2 - \frac{3}{2}(\mathbf{u} \cdot \mathbf{u})\right], \tag{3.14}$$

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

Utilizando essa aproximação chega-se na equação que governa a dinâmica do método de *lattice* Boltzmann:

$$f_i(\mathbf{x} + \mathbf{e}_i \Delta x, t + \Delta t) - f_i(\mathbf{x}, t) = \frac{1}{\tau}(f_i^{eq}(\mathbf{x}, \mathbf{v}, t) - f_i(\mathbf{x}, \mathbf{v}, t)), \tag{3.15}$$

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

3.4 Modelos de *lattice*

Vários modelos de *lattice* 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, por exemplo, o 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 (3.14) são:

$$w_0 = \frac{4}{6}, \;\; w_2 = \frac{1}{6}, \;\; w_3 = \frac{1}{6}. \tag{3.16}$$

Em duas dimensões existem vários modelos como o D2Q5, que possui cinco funções de distribuição de partículas. Assim as partículas de um nó podem se deslocar para um dos quatro nós vizinhos ou podem permanecer em repouso. Os pesos w_i possuem os seguintes valores:

$$w_0 = \frac{2}{6}, \;\; w_1 = \frac{1}{6}, \;\; w_2 = \frac{1}{6}, \;\; w_3 = \frac{1}{6}, \;\; w_4 = \frac{1}{6}. \tag{3.17}$$

O modelo D2Q9 possui oito direções de velocidade mais a probabilidade de permanecer em repouso em um nó. Este é o modelo mais comum para duas dimensões em simulações de escoamentos de fluidos. Os pesos w_i para este modelo são:

$$w_0 = \frac{4}{9}, \;\; w_{1-4} = \frac{1}{9}, \;\; w_{5-8} = \frac{1}{36}. \tag{3.18}$$

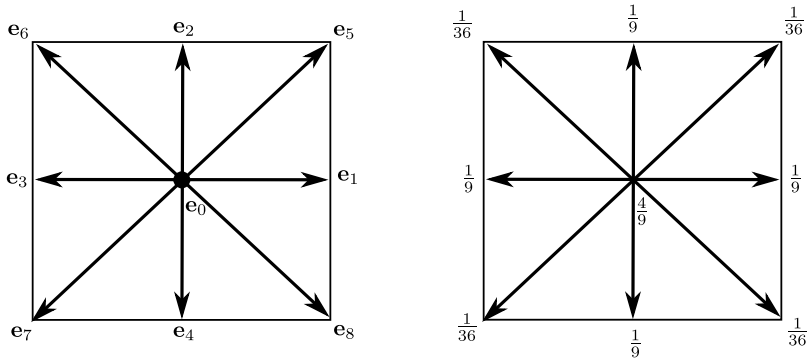


Figura 3.1: Modelo de *lattice* D2Q9 e seus coeficientes em cada direção.

A Figura 3.1 mostra a estrutura do modelo D2Q9. As variáveis macroscópicas do

fluido, densidade e velocidade, no MLB, podem ser recuperadas através das equações:

$$\rho = \sum_{i=0}^{8} f_i, \quad \rho\mathbf{u} = \sum_{i=0}^{8} e_i f_i. \tag{3.19}$$

Neste trabalho, o modelo de *lattice* D3Q7 (Yoshida e Nagaoka, 2010) foi usado para resolver o problema tridimensional de reação-difusão, considerando 7 funções de distribuição de partículas. Os coeficientes w_i para o modelo D3Q7 associados com a função de distribuição na direção $\mathbf{e}_i$ são dados por

$$w_0 = \frac{1}{4}, \quad w_{1-6} = \frac{1}{8}, \tag{3.20}$$

e as correspondentes direções de velocidade são apresentadas na Figura 3.2. Cada nó pode distribuir partículas para cada um dos seis nós vizinhos ou deixar partículas em repouso. As cores azul e vermelho das setas foram usadas para enfatizar direções opostas do *lattice*, que serão exploradas pelo algoritmo otimizado da propagação usado neste trabalho, que será detalhado posteriormente. Detalhes para um problema bidimensional de difusão usando o modelo D2Q5 podem ser encontrados em Yoshida e Nagaoka (2010).

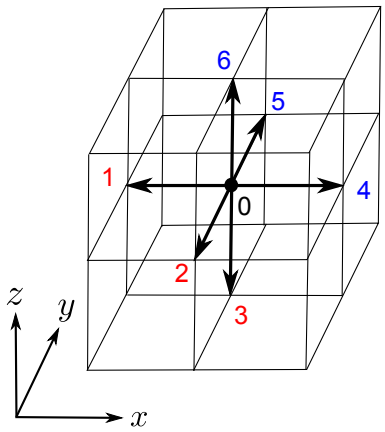


Figura 3.2: Modelo de *lattice* D3Q7 utilizado para resolver problemas de reação-difusão em três dimensões com sete direções de velocidade.

3.5 MLB para equações de reação-difusão

Para resolver o modelo de reação-difusão descrito pelo problema (2.19), que descreve a atividade elétrica no tecido cardíaco foi utilizado o método de *lattice* Boltzmann. Nesta

seção será descrito o MLB adaptado para simulação numérica de equações de reação-difusão, como discutido em Dawson *et al.* (1992) e Blaak e Sloot (2000), mas neste trabalho o foco são as equações da eletrofisiologia cardíaca.

Para simulações usando o MLB, o domínio é discretizado em uma malha igualmente espaçada. Todo nó da malha tem N direções de velocidade $\mathbf{e}_i$ $(i = 0, \ldots, N-1)$ e N funções de distribuição de partículas f_i, que descrevem a probabilidade de um certo número de partículas se deslocarem na direção de velocidade $\mathbf{e}_i$. No MLB, as funções de distribuição de partículas da quantidade escalar v no tempo t, posição $\mathbf{x}$ com velocidade na direção $\mathbf{e}_i$ é denotado por $f_i(\mathbf{x}, t)$. Neste caso, v é o potencial transmembrânico.

A equação de *lattice* Boltzmann para $f_i(\mathbf{x}, t)$ pode ser escrita como:

$$f_i(\mathbf{x} + \mathbf{e}_i \Delta x, t + \Delta t) - f_i(\mathbf{x}, t) = \Omega(\mathbf{x}, t), \tag{3.21}$$

onde $\Omega(\mathbf{x}, t)$ é o operador de colisão para v e depende das funções de distribuição das partículas $f_i(\mathbf{x}, t)$, Δt e Δx são o passo de tempo e o espaçamento da malha, respectivamente.

As quantidades macroscópicas são recuperadas calculando os momentos das funções de distribuição f_i. Assim, a variável macroscópica v, que é a solução do problema (2.19) pode ser recuperada fazendo a soma das distribuições f_i, como

$$v(\mathbf{x}, t) = \sum_{i=0}^{N-1} f_i(\mathbf{x}, t), \tag{3.22}$$

onde N é o número de direções do *lattice* utilizado.

Para problemas de reação-difusão, como é o caso da equação do monodomínio, o termo de colisão pode ser escrito como a soma de um termo reativo Ω^R e um termo não reativo denotado por Ω^{NR}, como descrito por Dawson *et al.* (1992). A parte não reativa do operador de colisão é descrita pela tradicional aproximação BGK, que é dada por

$$\Omega^{NR}(\mathbf{x}, t) = -\frac{1}{\tau}(f_i(\mathbf{x}, t) - f_i^{\mathrm{eq}}(\mathbf{x}, t)), \tag{3.23}$$

onde τ é o parâmetro de relaxação e f_i^{eq} é a função de distribuição de equilíbrio que depende da variável v e velocidade $\mathbf{u}$. Em geral, quando o MLB é aplicado para simulações

numéricas de fluidos, a função de distribuição de equilíbrio f_i^{eq} é dada por

$$f_i^{\text{eq}}(\mathbf{x},t) = w_i\ v\left(1 + \frac{(\mathbf{e}_i \cdot \mathbf{u})}{c_s^2} + \frac{(\mathbf{e}_i \cdot \mathbf{u})^2}{2c_s^4} - \frac{\mathbf{u}}{2c_s^2}\right), \tag{3.24}$$

onde c_s é a velocidade do som no *lattice* e w_i são os pesos para cada direção de velocidade, que dependem do tipo de *lattice* em uso. Devido ao interesse do presente trabalho ser voltado para a simulação de um problema de reação-difusão, a velocidade foi considerada como zero, ou seja, $\mathbf{u} = 0$. Neste caso, a função de distribuição de equilíbrio se reduz à seguinte forma

$$f_i^{\text{eq}}(\mathbf{x},t) = w_i\ v. \tag{3.25}$$

Para a parte reativa do operador de colisão, é assumido que R representa a mudança em v devido à reação (Dawson *et al.*, 1992), que permite a parte reativa ser distribuída entre as diferentes direções proporcionalmente aos coeficientes w_i,

$$\Omega^R(\mathbf{x},t) = w_i R. \tag{3.26}$$

3.5.1 MLB para o modelo do monodomínio

Na resolução do problema do monodomínio, o termo de reação R que aparece no operador de colisão Ω^R é determinado pelo modelo celular utilizado para descrever a cinética da membrana celular. Em geral, estes modelos são descritos por um sistema de EDOs não lineares. Em particular, para o modelo Luo-Rudy o termo reativo é $R = I_{ion} + I_{stim}$ onde I_{ion} é dado por (2.26) e I_{stim} é uma corrente de estímulo aplicada. Para o modelo TT2, o termo R é dado pela equação (2.29) somada a uma corrente de estímulo.

3.6 MLB para equação de reação-difusão anisotrópica

Os modelos de *lattice* Boltzmann mais utilizados para reproduzir o fenômeno de difusão são limitados à difusão isotrópica. A limitação existe devido à utilização da aproximação BGK para o operador de colisão, que não tem parâmetros suficientes para descrever uma

difusão anisotrópica. Esta aproximação é baseada em um único parâmetro de relaxação no tempo τ, como mostrado na equação (3.23) e, portanto, este modelo é geralmente conhecido como *single-relaxation-time* (SRT).

Com o objetivo de contornar esta limitação, alguns estudos, como aqueles realizados por Zhang *et al.* (2002), Ginzburg (2005) e Yoshida e Nagaoka (2010), propuseram diferentes métodos para incorporar a difusão anisotrópica no MLB. Yoshida e Nagaoka (2010) propuseram em seu trabalho o uso do operador de colisão conhecido como *multiple-relaxation-time* (MRT) para considerar a anisotropia em um problema de advecção-difusão. No presente trabalho também foi utilizado o operador de colisão MRT para considerar a anisotropia do tecido cardíaco no problema de reação-difusão.

Sejam $\mathbf{f}$ e $\mathbf{f}^{\text{eq}}$ os vetores coluna formados pelas funções de distribuição f_i e funções de distribuição de equilíbrio f_i^{eq}. Note que o modelo SRT pode ser escrito em termos do operador de colisão como

$$\Omega^{NR} = -\mathbf{A}(\mathbf{f} - \mathbf{f}^{\text{eq}}), \tag{3.27}$$

onde a matriz de colisão $\mathbf{A}$ é definida em termos de τ como $\mathbf{A} = (1/\tau)\mathbf{I}$, onde $\mathbf{I}$ é a matriz identidade.

O modelo MRT projeta o vetor $\mathbf{f}$ em um espaço vetorial onde cada componente corresponde a um determinado momento de $\mathbf{f}$. Então, o operador de colisão é aplicado no espaço de momento a fim de relaxar cada componente para o equilíbrio com diferentes parâmetros de relaxação. Finalmente, o vetor é projetado de volta ao espaço original. Assim, o MRT permite ajustar o coeficiente de relaxação para cada momento separadamente, possibilitando considerar o efeito da difusão anisotrópica.

De acordo com Yoshida e Nagaoka (2010), o modelo MRT é dado por

$$\Omega^{NR} = -\mathbf{M}^{-1}\mathbf{S}\mathbf{M}(\mathbf{f} - \mathbf{f}^{\text{eq}}), \tag{3.28}$$

onde $\mathbf{M}$ é a matriz que projeta um vetor no espaço de momentos. A definição da matriz $\mathbf{M}$ depende da escolha dos momentos. O primeiro momento foi considerado como a quantidade conservada, isto é, o potencial transmembrânico v. O segundo, terceiro e quarto momentos são relacionados com o fluxo de v na direção i e os momentos restantes foram obtidos usando o procedimento de Gram-Schmidt (Trefethen e Bau, 1997). A

matriz $\mathbf{M}$ e a matriz de relaxação no tempo $\mathbf{S}$ são definidas, respectivamente, por

$$\mathbf{M} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 \\ 6 & -1 & -1 & -1 & -1 & -1 & -1 \\ 0 & 2 & 2 & -1 & -1 & -1 & -1 \\ 0 & 0 & 0 & 1 & 1 & -1 & -1 \end{bmatrix}, \quad \mathbf{S}^{-1} = \begin{bmatrix} \tau_0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \bar{\tau}_{xx} & \bar{\tau}_{xy} & \bar{\tau}_{xz} & 0 & 0 & 0 \\ 0 & \bar{\tau}_{yx} & \bar{\tau}_{yy} & \bar{\tau}_{yz} & 0 & 0 & 0 \\ 0 & \bar{\tau}_{zx} & \bar{\tau}_{zy} & \bar{\tau}_{zz} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \tau_4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \tau_5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \tau_6 \end{bmatrix}.$$

Aqui τ_0, τ_4, τ_5, τ_6 e $\bar{\tau}_{ij}$ são os coeficientes de relaxação no tempo. Em Yoshida e Nagaoka (2010) é apresentado que a relação entre os coeficientes de relaxação e as componentes do tensor de condutividade σ_{ij} é dada por

$$\bar{\tau}_{ij} = \frac{1}{2}\delta_{ij} + 4\frac{\Delta t}{\Delta x^2}\sigma_{ij}. \tag{3.29}$$

O coeficiente τ_0 é relacionado com o potencial, já τ_4, τ_5 e τ_6, que são relacionados aos momentos de mais alta ordem, não afetam diretamente a difusão, mas afetam a estabilidade do método (Yoshida e Nagaoka, 2010).

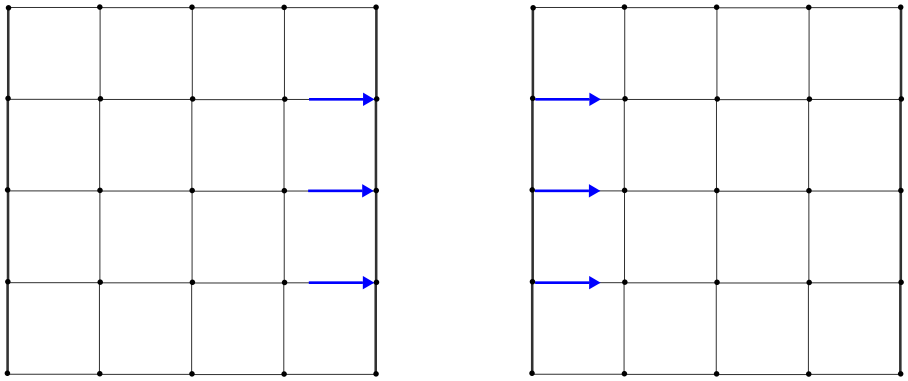
3.7 Condições de contorno

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, porém, nesta seção, será apresentado apenas o tratamento da condição de contorno periódica e da condição de fluxo nulo.

3.7.1 Periódica

Este tipo de condição de contorno é usada quando busca-se simular uma parte de um domínio infinito ou muito grande, onde os efeitos do contorno são ignorados. Esta condição de contorno é de fácil entendimento e implementação, ela conecta duas regiões do contorno. Esta condição é imposta colocando as distribuições que estão saindo do domínio por um lado do contorno como entrada na parte oposta a este contorno. A Figura 3.3 exemplifica

esta condição, onde as as distribuições que estão saindo pelo contorno da borda direita, Figura 3.3a, são colocadas como entrada na borda direita, Figura 3.3b.



(a) (b)

Figura 3.3: Tratamento do contorno tipo periódica. (a) Distribuições antes do tratamento do contorno e (b) após tratamento.

3.7.2 Condição do tipo fluxo nulo

As condições de fluxo nulo para o potencial transmembrânico v são do tipo Neumann homogênea. Este tipo de condição de contorno pode ser facilmente implementada no método de *lattice* Boltzmann através da condição *bounce-back* (Succi, 2013), também chamada de condição de parede no contexto de simulações de fluidos. Como o próprio nome diz, o contorno é considerado como uma parede, portanto toda partícula que chega ao contorno continua se deslocando na mesma direção, mas tem seu sentido de velocidade invertido. Então, a condição é imposta fazendo com que a função de distribuição que chega a um nó do contorno se deslocando para fora do domínio seja atribuída à função de distribuição do mesmo nó que tenha mesma direção e sentido oposto.

Para descrever esta estratégia, considere um domínio bidimensional e o modelo de *lattice* D2Q5. Suponha um nó localizado na aresta esquerda do domínio da Figura 3.4, isto é, $\mathbf{x} = (0, y)$, onde a função de distribuição f_2 chegou ao nó e está se propagando para fora do domínio. Já a função de distribuição f_1, representada por uma seta tracejada, é desconhecida neste nó, pois não existem funções de distribuição se propagando de fora do domínio para o contorno. Aplicando a condição de contorno *bounce-back*, f_2 é atribuída à f_1, caracterizando uma condição de fluxo nulo no contorno. Nesse exemplo, a condição de contorno deve ser aplicada para todos os nós deste contorno e pode ser expressa por

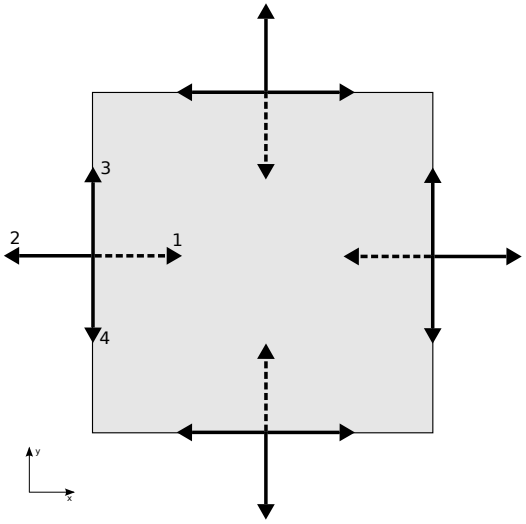
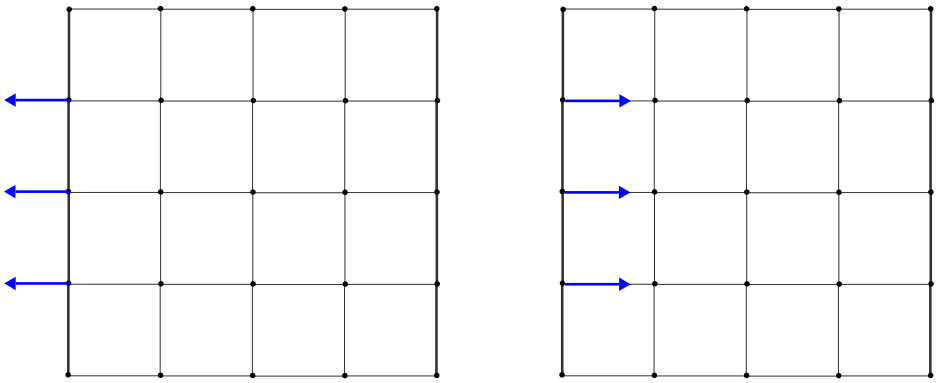


Figura 3.4: Funções de distribuição no contorno em um *lattice* D2Q5. Setas tracejadas indicam funções de distribuição desconhecidas no contorno.

$f_1(\mathbf{x}, t + \Delta t) = f_2(\mathbf{x}, t)$. Assim, quando uma distribuição chega à um nó do contorno, saindo do domínio, seu sentido é invertido, como mostrado na Figura 3.5.



(a) (b)

Figura 3.5: Tratamento do contorno tipo *bounce-back*. (a) Distribuições chegando ao contorno e (b) Distribuições no contorno após tratamento.

Em geral, pode-se expressar este tratamento de contorno como

$$f_a(\mathbf{x}, t + \Delta t) = f_b^*(\mathbf{x}, t), \tag{3.30}$$

onde f_b^* é a função de distribuição após colisão e o índice b indica a direção oposta ao a, ou seja, $\mathbf{e}_a = -\mathbf{e}_b$.

3.8 Análise multiescala

É possível mostrar que o MLB para simulação de escoamento de fluidos recupera 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 verificados nos trabalhos de Wolf-Gladrow (2000) e Golbert (2009). Existem também trabalhos como Dawson *et al.* (1992) e Blaak e Sloot (2000) que apresentam a análise multiescala para problemas de reação-difusão e para problemas de difusão-advecção (Yoshida e Nagaoka, 2010). Este trabalho é voltado para a aplicação do método de *lattice* Boltzmann aos problemas de reação-difusão, não tendo como foco a análise de erro entre o MLB e as equações de Navier-Stokes.

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. (3.15) 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 $\mathrm{O}\big(Ma^3\big)$ na equação do momento, isto é:

$$\nabla \cdot \mathbf{u} = 0 + \mathrm{O}\big(Ma^2\big) \tag{3.31}$$

$$\rho \frac{\partial \mathbf{u}}{\partial t} = \rho \mathbf{g} - \nabla p + \nu \nabla^2 \cdot \mathbf{u} - \mathbf{u} \cdot \nabla \mathbf{u} + \mathrm{O}\big(Ma^3\big), \tag{3.32}$$

que junto às condições de contorno apropriadas completam a descrição do modelo para simulação de fluidos. Através da expansão multiescala de Chapman-Enskog, é possível mostrar ainda a seguinte relação entre a viscosidade do fluido e o parâmetro de relaxação:

$$\nu = \frac{\Delta x^2}{3\Delta t}\left(\frac{1}{\tau} - \frac{1}{2}\right). \tag{3.33}$$

Ao realizar uma análise assintótica no MLB para problemas de reação-difusão é possível recuperar a equação que modela este problema. Yoshida e Nagaoka (2010) realizou esta análise para um problema de advecção-difusão, obtendo a seguinte relação entre os parâmetros de relaxação e os coeficientes de difusão:

$$\bar{\tau}_{ij} = \frac{1}{2}\delta_{ij} + 4\frac{\Delta t}{\Delta x^2}\sigma_{ij}. \tag{3.34}$$

onde $\bar{\tau}_{ij}$ são os parâmetros de relaxação considerados e δ_{ij} é o delta de Kronecker.

4 Fundamentos de programação paralela

A simulação de problemas de interesse científico tem demandado cada vez mais poder computacional, para que estes sejam resolvidos o mais rápido possível ou para abordar problemas cada vez maiores. 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 a ferramenta utilizada para desenvolver as versões paralelas das implementações deste trabalho, CUDA (NVIDIA, 2013).

4.1 Arquiteturas de Computação Paralela

As duas principais arquiteturas disponíveis para programação paralela são os sistemas de memória compartilhada e os sistemas de memória distribuídas, que necessitam de estratégias diferentes para a implementação.

4.1.1 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 4.1. Devido a este compartilhamento é necessária uma supervisão para que dois processadores diferentes escrevam no mesmo endereço de memória. A biblioteca OpenMP (Quinn, 2003) fornece recursos para programação paralela segundo

a arquitetura de memória compartilhada. A plataforma CUDA (NVIDIA, 2013), que descrevemos a seguir, também é uma plataforma que utiliza a arquitetura de memória compartilhada.

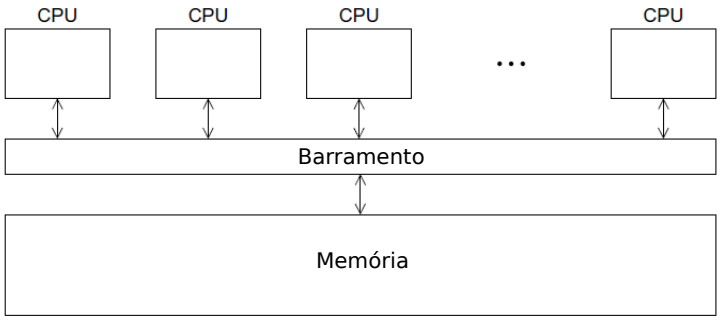


Figura 4.1: Sistema de memória compartilhada. Adaptado de Pacheco (2011).

Um único sistema operacional gerencia os recursos disponíveis e os processadores que possuem vários núcleos, tem seus núcleos, tratados como unidades de processamento individuais, onde todos estes têm 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. Entretanto, 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.

4.1.2 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 mostra a Figura 4.2. 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 pela

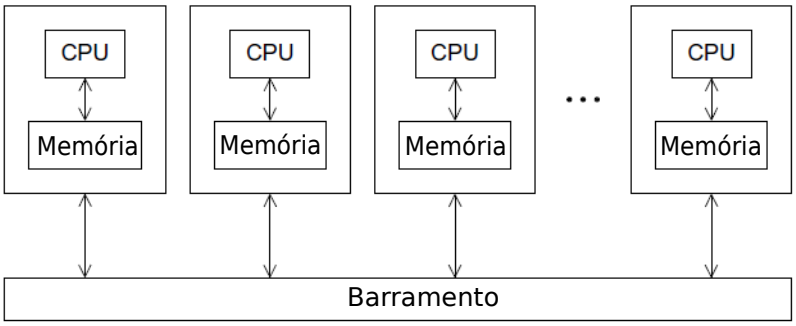


Figura 4.2: Sistema de memória distribuída. Adaptado de Pacheco (2011)

troca de mensagens na rede. Então a rede de comunicação se torna uma parte importante do sistema, pois quanto mais lenta a rede mais tempo será gasto com comunicação e esse tempo 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.

4.2 Métricas de desempenho

Uma forma de medir o desempenho de uma aplicação paralela 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,

$$S = \frac{T_{sequencial}}{T_{paralelo}}. \tag{4.1}$$

O desempenho ideal de uma aplicação paralela seria linear, isto é, o *speedup* é igual ao número de processadores. Mas na prática é muito difícil de se conseguir tal desempenho, pois muito tempo pode ser gasto na comunicação entre processos, ou em sua sincronização, entre outros problemas. O *speedup* também pode ser super linear, que acontece quando o problema tem pouca comunicação e ainda aproveita do bom uso da memória cache, evitando a busca de dados na memória principal, que é mais lenta.

4.2.1 MLUPS

A métrica MLUPS (*Mega Lattice Updates Per Second*) é bastante utilizada para reportar o desempenho de implementações do método de *lattice* Boltzmann, portanto, foi utilizada neste trabalho para analisar o desempenho da implementação realizada. Esta métrica indica quantos elementos de *lattice* foram atualizados em um segundo e é calculado da seguinte forma:

$$MLUPS = \frac{N_{\text{nós}} \times N_{passos}}{T} \tag{4.2}$$

onde T é o tempo total da solução do MLB, $N_{\text{nós}}$ é o número de nós do *lattice* e N_{passos} é o número de iterações no tempo. Como o MLB foi usado apenas para resolver a EDP parabólica (2.19), T foi tomado como o tempo gasto na solução da equação parabólica do monodomínio.

4.2.1.1 Modelo de desempenho

Um modelo de desempenho pode ser usado para descobrir o número máximo de MLUPS que uma implementação executada em GPU pode alcançar (Habich *et al.*, 2011). Para implementações onde o acesso à memória é o gargalo, como no presente trabalho, o número de MLUPS máximo pode ser calculado por

$$MLUPS_{max} = \frac{BW}{n_{bytes}}, \tag{4.3}$$

onde BW é a largura de banda da memória e n_{bytes} é o número de *bytes* lidos ou escritos na memória para cada atualização de um nó do *lattice*, que pode ser calculado por

$$n_{bytes} = (n_{loads} + n_{store})s, \tag{4.4}$$

onde n_{loads} é o número de operações de leitura na memória, n_{store} é o número de operações de escrita na memória e s é o tamanho em *bytes* do tipo dados utilizado. Neste trabalho, $s = 4$ para precisão simples e $s = 8$ para precisão dupla.

4.3 GPGPU e CUDA

Foi necessário para o avanço de aplicações 3D, como jogos que exibem imagens cada vez mais próximas da realidade, 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 ser utilizadas na resolução de outros problemas, como aplicações científicas, surgindo o que se denomina de GPGPU (*General Purpose Graphics Processing Unit*). A Figura 4.3 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.

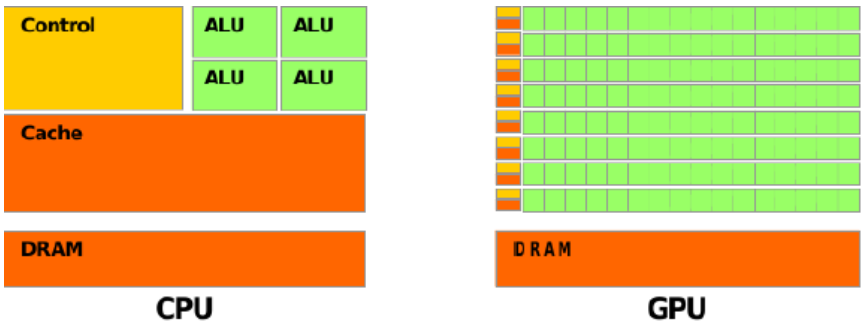


Figura 4.3: Comparação entre as arquiteturas da CPU e da GPU.

O objetivo da computação em GPU é executar qualquer tipo de algoritmo, não necessariamente relacionado ao processamento gráfico. As GPUs possuem processadores *multicores* altamente paralelos, *multithread* e com uma alta largura de banda de memória. 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 (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 que possui um modelo de programação paralela explícito. Na arquitetura CUDA existem um processador, chamado de hospedeiro (*host*) e uma GPU, chamada de dispositivo (*device*). Um código em CUDA começa a ser executado pelo hospedeiro, mas podem existir partes do código que precisam ser executadas pelo dispositivo. Então o hospedeiro faz chamadas ao dispositivo para executar estas partes do código. Para realizar sua execução, o dispositivo pode precisar de dados que estão na memória do hospedeiro, portanto, é necessário fazer operações de cópia da memória do hospedeiro para a memória do dispositivo. Em seguida, o dispositivo executa sua tarefa e ao fim pode enviar dados de volta para o hospedeiro, também através de cópia de memória.

4.3.1 Arquitetura CUDA

A arquitetura CUDA é construída como um vetor escalável de multiprocessadores, chamados *streaming multiprocessors* (SMs). Estes são constituídos de núcleos, chamados *scalar processors* (SPs), unidades funcionais e de uma memória compartilhada. Os SMs são responsáveis por criar, gerenciar e executar as *threads* em *hardware*.

O multiprocessador organiza as *threads* em grupos de tamanho 32, chamados de *warps*. Todos os SPs recebem a mesma instrução, podendo ou não executá-la em função do predicado associado à instrução. Assim, as *threads* podem ter diferentes caminhos de execução. A eficiência total é alcançada quando todas as *threads* de um *warp* possuem o mesmo caminho de execução. Se existem caminhos diferentes, o *warp* executa sequencialmente cada desvio tomado.

Para se executar um código na GPU deve ser criada uma função especial chamada de *kernel*. Quando invocada pela CPU, esta é executada na GPU em paralelo por diferentes *threads*. Cada *thread* que executa o *kernel* possui um identificador único e este pode ser acessado dentro do *kernel* através da variável *threadIdx*.

As *threads* são divididas em blocos e cada bloco é executado em um multiprocessador da GPU. Existe um limite de *threads* por bloco, devido ao fato de que as *threads* de um

bloco são executadas em um mesmo multiprocessador e devem compartilhar os recursos de memória, que são limitados. As GPUs atuais possuem um limite de 1024 *threads* por bloco. Os blocos também possuem um identificador único, que pode ser acessado através da variável *blockIdx* e a dimensão do bloco pode ser acessada pela variável *blockDim*. Um conjunto de blocos é chamado de *grid*, podendo ter até três dimensões.

Quando a CPU chama uma função *kernel*, ela deve especificar o tamanho do bloco e o tamanho do *grid*, através do operador $<<< \cdots >>>$, onde cada *thread* geralmente se refere a um dado do problema. As *threads* podem acessar dados de diferentes espaços da memória. Cada *thread* tem uma memória local privada, as *threads* de um mesmo bloco possuem uma memória compartilhada e visível por todas que pertencem ao mesmo bloco. Além disso, todas as *threads* tem acesso à memória global. Existem ainda memórias do tipo somente-leitura, que são a memória constante e a memória de textura.

4.3.2 Organização da memória

A memória global reside na memória do dispositivo, permitindo leitura e escrita. Quando o acesso é feito de forma coalescente para todas as *threads* do *warp*, uma única instrução de memória é realizada. Caso contrário, várias instruções de memória devem ser executadas. O acesso à memória global é lento, portanto é necessário realizar o mínimo possível de operações de memória e uma das formas é garantir o acesso coalescente.

A memória compartilhada está localizada nos multiprocessadores, também pode ser usada para leitura e escrita. A memória compartilhada é alocada por bloco, portanto as *threads* do bloco podem acessar dados que foram carregados da memória global por outras *threads* do mesmo bloco. Além disso, o tempo de acesso à memória compartilhada é muito menor do que o acesso à memória global.

A memória constante, também chamada de cache constante, localiza-se na memória do dispositivo e possui $64KB$, além de ser somente-leitura, portanto deve ser usada para variáveis que não sofrerão alteração durante a execução do *kernel*. Esta memória é compartilhada por todas as *threads* e possui acesso mais rápido do que à memória global.

A memória de textura, ou cache de textura, reside na memória do dispositivo e também é somente-leitura. Este tipo de memória foi desenvolvido para aplicações gráficas, mas também pode ser útil para outros tipos de computações. Existem funções específicas para acessar este tipo de memória e o acesso pode ser melhorado, em relação à memória global,

quando *threads* de um mesmo *warp* acessam dados próximos na memória, tirando proveito da localidade espacial. Além disso, esta memória possui um tamanho de armazenamento maior do que a memória constante.

A GPU usada no presente trabalho foi a placa NVIDIA Tesla M2075 e algumas especificações desta GPU se encontram na Tabela 4.1. As informações da tabela foram obtidas usando os programas `deviceQuery` e `bandwidthTest`, presentes no SDK (*Software Development Kit*) da NVIDIA.

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;
- Desalocar memória da CPU e da GPU.

A grande perda de desempenho na programação em CUDA é devido às transações de memória, portanto deve-se reduzir ao máximo o acesso à memória global, que é muito lento, e buscar alocar os dados de forma que o acesso seja coalescente, diminuindo o número de transações de memória. Além disso, utilizar os outros tipos de memória sempre que possível, pois possuem um tempo de acesso menor do que à memória global. Outro problema está na transferência de dados entre hospedeiro e dispositivo, então deve-se evitar este tipo de transação ao máximo. Também ocorre perda de desempenho quando existe mais de um caminho de execução dentro do mesmo *warp*, pois se isso acontece as instruções são realizadas de forma sequencial, portanto deve-se evitar desvios condicionais. Atentando-se a esses detalhes, uma boa implementação em CUDA pode ser realizada, resultando em boas acelerações em relação à implementação sequencial.

Tabela 4.1: Especificações da GPU usada no presente trabalho, obtidas pelos programas `deviceQuery` e `bandwidthTest`, presentes no SDK da NVIDIA.

Tesla M2090	
Multiprocessadores	16
Núcleos	512
Clock da GPU	1.3 GHz
Memória global	6144 MB
Memória compartilhada por bloco	49152 B
Memória de constante	65536 B
Cache L2	786432 B
Largura de banda da memória	105430 MB/s

5 Implementação

Neste capítulo detalhes sobre a implementação do MLB para resolver a equação do monodomínio são apresentados. Primeiramente, será apresentada uma implementação geral do MLB e detalhes sobre a estrutura de dados utilizada, em seguida, uma otimização da etapa de propagação através do algoritmo *swap* será descrita. Então detalhes sobre a representação esparsa utilizada e sobre a implementação paralela desenvolvida serão apresentados.

5.1 Implementação Geral

A implementação do MLB pode ser dividida em:

- Inicialização, onde os parâmetros do método e a condições iniciais são aplicados;
- Colisão, onde o operador de colisão é aplicado, calculando as funções de distribuição em um determinado instante de tempo;
- Propagação, onde as funções de distribuição de um nó podem se propagar para os vizinhos;
- Contorno, onde é feito o tratamento das condições de contorno.

A implementação para o problema do monodomínio procede da seguinte forma: primeiro, dados v^n e $\boldsymbol{\eta}^n$, as aproximações atuais para v e $\boldsymbol{\eta}$ no tempo $t = t_n$, resolve-se o sistema de EDOs dado por

$$\begin{aligned} C_m \frac{dv}{dt} &= -I_{ion}(v, \boldsymbol{\eta}) + I_{stim}, \quad & v(t_n) &= v^n, \\ \frac{d\boldsymbol{\eta}}{dt} &= \mathbf{f}(v, \boldsymbol{\eta}), & \boldsymbol{\eta}(t_n) &= \boldsymbol{\eta}^n, \end{aligned} \tag{5.1}$$

onde o termo I_{stim} denota um estímulo externo que é geralmente aplicado com objetivo de iniciar a atividade elétrica de miócitos cardíacos. Utiliza-se o termo $R = I_{ion}(v^n, \boldsymbol{\eta}^n) + I_{stim}$ para calcular as funções de distribuição de equilíbrio do MLB durante a etapa de colisão. Em seguida, são realizadas as etapas de propagação e o tratamento das condições de contorno para obter a solução aproximada v^{n+1} no tempo $t = t_n + \Delta t$.

A solução numérica do MLB aplicado ao problema de reação-difusão da eletrofisiologia cardíaca é apresentada no Algoritmo 1. Inicialmente, os parâmetros do método e do modelo celular são iniciados, em seguida os vetores do MLB e das EDOs são alocados e iniciados com seus valores iniciais, isto é, todos os nós do *grid* são inicializados com os valores da condição inicial do modelo celular adotado (linhas 1-4). Em seguida, dados v_n e $\boldsymbol{\eta}^n$ do passo de tempo anterior, as variáveis de estado $\boldsymbol{\eta}$ são atualizadas utilizando o método de Rush-Larsen (linhas 6 e 7). O próximo passo é o *loop* no tempo, onde a cada passo as etapas de colisão (linhas 8-13), propagação (linhas 14-16) e condição de contorno (linhas 17 e 18) do MLB são aplicadas. Ao final, o valor do potencial transmembrânico é armazenado em arquivo para posterior visualização. As variáveis auxiliares $\hat{\mathrm{f}}_i$ foram utilizadas para facilitar a implementação da etapa de propagação, onde as distribuições podem se deslocar de um ponto para outro. A utilização destas variáveis auxiliares $\hat{\mathrm{f}}_i$ facilitam a implementação, mas aumentam o consumo de memória. Uma alternativa é a utilização do algoritmo *swap* para a propagação, que será apresentado posteriormente.

5.1.1 Estrutura de dados

Na resolução do modelo monodomínio, a cada passo de tempo, o valor do potencial transmembrânico e das variáveis de estado de todos os nós considerados na malha são lidos e alterados. Esses dados podem ser armazenados criando um *array* de dimensão `Nx*Ny*Nz*Nv` organizado como: *array* de estruturas ou estrutura de *arrays*, onde `Nv` é o número de variáveis da estrutura, `Nx`, `Ny` e `Nz` são o número de nós nas direções x, y e z, respectivamente. A organização em *array* de estruturas cria um vetor onde a estrutura de cada nó é armazenada sequencialmente, como mostra a Figura 5.1a, onde as cores representam variáveis distintas. Já na representação em estrutura de *arrays* a primeira variável da estrutura de todos os nós são armazenadas sequencialmente, em seguida, a segunda variável da estrutura de todos os nós é armazenada e assim para todas as variáveis presentes na estrutura, como pode ser visto na Figura 5.1b.

Para obter uma implementação GPU otimizada, o formato de alocação de memória para as funções de distribuição das partículas foi escolhido cuidadosamente. Para otimizar transações de memória, as funções de distribuição foram armazenadas na memória usando um vetor unidimensional de tal forma que um nó do *lattice* possa ser acessado como `f[i*Nx*Ny*Nz+z*Ny*Nx+y*Nx+x]`, onde `i` é o índice da função de distribuição que será

Algoritmo 1: Algoritmo do MLB aplicado ao problema do monodomínio

Configuração dos parâmetros do modelo
Inicialização das EDOs
para cada *nó* **x faça**
    Inicialização do MLB: $f_i = w_i v^0(\mathbf{x}),\ \ i = 0, \cdots, 6$
para $k \leftarrow 1$ **até** *numIteracoes* **faça**
    para cada *nó* **x faça**
        Calcula o termo de reação $R = I_{ion}(v^n, \boldsymbol{\eta}^n) + I_{stim}$, e as variáveis de estado $\boldsymbol{\eta}^{n+1}(v^n, \boldsymbol{\eta}^n, \mathbf{x})$, usando o método Rush-Larsen
    Colisão:
    para cada *nó* **x faça**
        Calcula as distribuições de equilíbrio: $f_i^{eq} = w_i v^n(\mathbf{x}),\ \ i = 0, \cdots, 6$
        Calcula o operador de colisão: $\boldsymbol{\Omega}(\mathbf{x}, t) = -\mathbf{M}^{-1}\mathbf{S}\mathbf{M}(\mathbf{f} - \mathbf{f^{eq}}) + \mathbf{w}\Delta t R$
        Atualiza as distribuições: $f_i^{tmp}(\mathbf{x}, t) = f_i(\mathbf{x}, t) + \Omega_i(\mathbf{x}, t),\ \ i = 0, \cdots, 6$
        Calcula a variável macroscópica: $v^{n+1}(\mathbf{x}) = \sum_{i=0}^{6} f_i^{tmp}(\mathbf{x}, t)$
    Propagação:
    para cada *nó* **x faça**
        $f_i(\mathbf{x} + \mathbf{e}_i, t + \Delta t) = f_i^{tmp}(\mathbf{x}, t),\ \ i = 0, \cdots, 6$
    para cada *nó do contorno* **faça**
        Condição de contorno usando Eq. (3.30)
    Grava dados em arquivo
    $t \leftarrow t + \Delta t$
    $k \leftarrow k + 1$

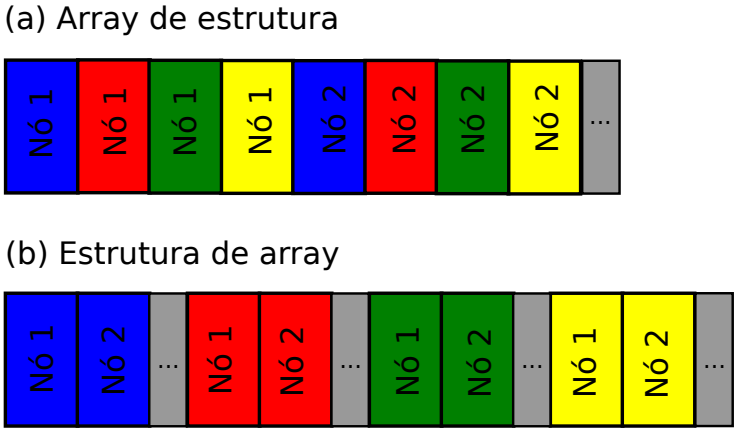


Figura 5.1: Representação de um *array* de estruturas e uma estrutura de *arrays*.

acessada, `x`, `y`, `z` são os índices nas direções x, y e z, respectivamente. Assim a ordem dos dados seguem o formato de estrutura de *arrays* na implementação em CUDA, apresentado na Figura 5.1b que possibilita o acesso à memória de maneira coalescente.

5.2 Algoritmo Swap

Implementações tradicionais do MLB geralmente usam um vetor temporário adicional para armazenar as funções de distribuição de partículas f_i^{tmp} após a colisão, que durante a etapa de propagação se deslocam no *lattice*. Caso contrário, haverá sobrescrita de dados durante a propagação. A propagação, representada pelas linhas 14, 15 e 16 do Algoritmo 1, pode ser realizada para o modelo de *lattice* D3Q7 pelo Algoritmo 2, que detalha a implementação usual da etapa de propagação. Nesta etapa, as distribuições de cada nó do *lattice*, que acabaram de sofrer colisão, propagam-se para um dos nós vizinhos. No contorno algumas distribuições podem ter índices inválidos, que devem ser tratados.

Algoritmo 2: Algoritmo da etapa de propagação do MLB para o *lattice* D3Q7.

```
1  para i ← 0 até Nx faça
2      para j ← 0 até Ny faça
3          para k ← 0 até Nz faça
4              f_0[i,j,k] = f_0^tmp[i,j,k]
5              f_1[i,j,k] = f_1^tmp[i+1,j,k]
6              f_2[i,j,k] = f_2^tmp[i,j+1,k]
7              f_3[i,j,k] = f_3^tmp[i,j,k+1]
8              f_4[i,j,k] = f_4^tmp[i-1,j,k]
9              f_5[i,j,k] = f_5^tmp[i,j-1,k]
10             f_6[i,j,k] = f_6^tmp[i,j,k-1]
```

Uma estratégia para contornar este uso de memória extra é o algoritmo de *swap* (Mattila *et al.*, 2007; Latt, 2007) para a etapa de propagação, que faz troca de posições no vetor de funções de distribuição e não copia dados para um vetor temporário. Este algoritmo requer que os índices das funções de distribuição das partículas sejam organizados de maneira que a função de distribuição que considera partículas em repouso tenha índice 0 e que funções de distribuição opostas possam ser calculadas por

$$oposta(i) = i + (N-1)/2, \quad i = 1 \ldots (N-1)/2. \tag{5.2}$$

A organização da funções de distribuição para o *lattice* utilizado pode ser vista na Figura 3.2. O algoritmo requer uma ordem de execução dos nós para que a propagação seja feita de forma correta e procede continuamente trocando metade das funções de

distribuição de um nó com seus vizinhos.

Para facilitar o entendimento do algoritmo *swap*, sem perda de generalidade, ele será apresentado para um modelo de *lattice* D2Q5. Neste algoritmo a propagação é dividida em duas etapas, onde primeiro são realizadas trocas de distribuições de um mesmo nó e depois são feitas trocas de distribuições com nós vizinhos. Após a etapa de colisão ser realizada, o *lattice* se encontra como na Figura 5.2, as cores diferentes em cada nó indicam funções de distribuição opostas.

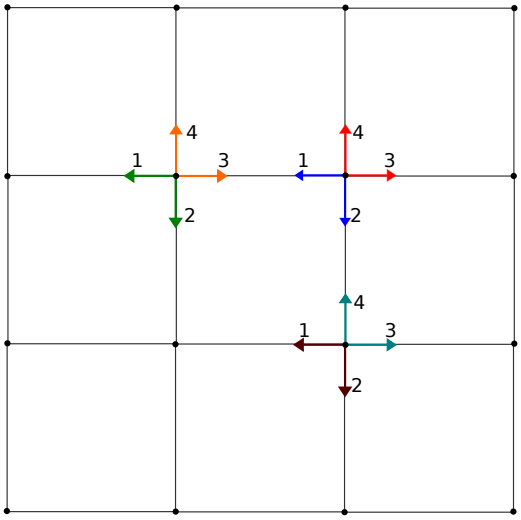
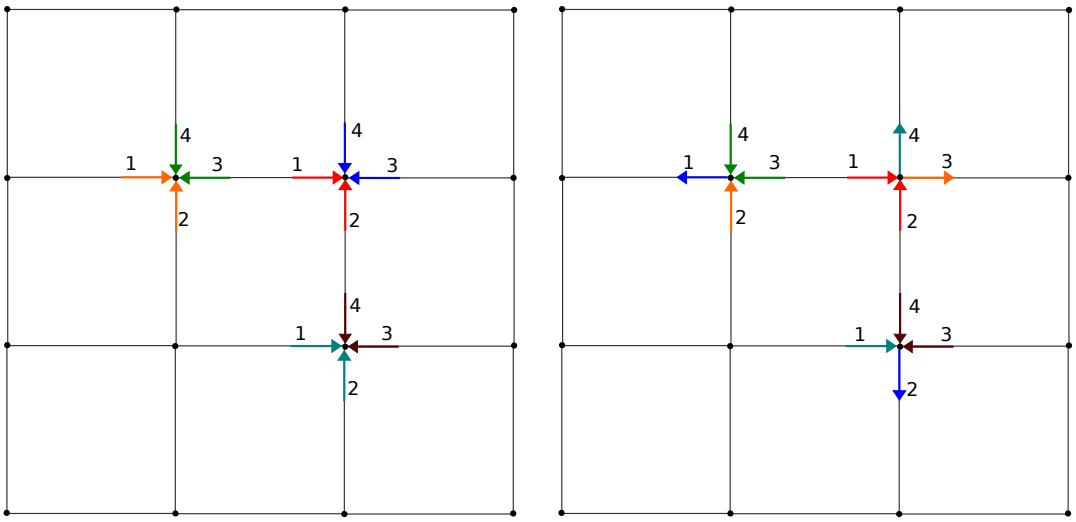


Figura 5.2: Funções de distribuição antes da etapa de propagação.

A primeira parte da propagação troca distribuições opostas localmente, como mostrado na Figura 5.3a. Pode-se observar que para realizar estas trocas apenas metade das distribuições são acessadas na memória. Antes de executar a segunda parte da propagação, a primeira etapa deve ter sido executada para todos os nós do *lattice*.

Finalizada a primeira parte da propagação *swap*, pode-se realizar a segunda que consiste em trocar funções de distribuição de um nó por distribuições de nós vizinhos. Então cada nó troca metade de suas distribuições por distribuições opostas de um nó vizinho, como na Figura 5.3b. Considerando o nó que possui as distribuições nas cores azul e vermelha, pode-se observar que a distribuição $f_3 = oposta(1)$ é trocada pela distribuição f_1 do nó à esquerda e a distribuição $f_4 = oposta(2)$ é trocada por f_2 do nó abaixo. A outra metade das distribuições do nó serão trocadas quando o mesmo procedimento for aplicado em um nó vizinho. Assim ao realizar a segunda etapa em todos nós, todas as distribuições terão sido propagadas.



(a) (b)

Figura 5.3: Etapas do Algoritmo *swap*. (a)Funções de distribuição após primeira etapa da propagação, onde as distribuições opostas são trocadas localmente. (b) Funções de distribuição após segunda etapa da propagação, que troca a distribuição 3 pela distribuição laranja do nó à esquerda e troca a distribuição 4 pela distribuição azul claro do nó abaixo.

Com o objetivo de diminuir o acesso à memória, o algoritmo *swap* pode ser melhorado, de forma a acessar as funções de distribuição uma única vez. Além disso, com esta abordagem pode-se juntar as etapas de colisão e propagação, fazendo um único *loop* para realizar todo o processo, como mostra o Algoritmo 3. Para fazer isto, é necessária uma troca envolvendo três distribuições. Considerando o nó que possui as distribuições nas cores azul e vermelha da Figura 5.2, primeiramente guarda-se a distribuição f_1 em uma variável temporária (linha 13 do Algoritmo 3), em seguida coloca-se a distribuição oposta no lugar de f_1 (linha 14). Então armazena-se no lugar da distribuição oposta à f_1 a distribuição que se encontra na posição para onde f_1 vai se propagar (linha 15). Por fim, f_1 vai para a posição onde ela se propagaria (linha 16). O mesmo procedimento é feito para a distribuição f_2 e em seguida repete-se o processo para todos os outros nós.

Ao final da segunda etapa da propagação o *lattice* estará consistente e pronto para o próximo passo do MLB. Com este algoritmo não existe sobrescrita de dados e também não há dependência entre nós, o que possibilita que as etapas da propagação sejam feitas em todos os nós ao mesmo tempo, sendo atrativo para implementações paralelas. Além disso, o tratamento da condição de contorno de fluxo nulo já é realizado automaticamente pelo algoritmo *swap* (Latt, 2007).

Algoritmo 3: Algoritmo que realiza as etapas de colisão e propagação juntas, utilizando a estratégia *swap*.

para cada *nó* **x faça**
  Colisão:
  Calcula as distribuições de equilíbrio: $f_i^{eq} = w_i v^n(\mathbf{x}),\ \ i = 0, \cdots, N-1$
  Calcula o operador de colisão: $\mathbf{\Omega}(\mathbf{x}, t) = -\mathbf{M}^{-1}\mathbf{S}\mathbf{M}(\mathbf{f} - \mathbf{f^{eq}}) + \mathbf{w}\Delta t R$
  Atualiza as distribuições: $f_i(\mathbf{x}, t) = f_i(\mathbf{x}, t) + \Omega_i(\mathbf{x}, t),\ \ i = 0, \cdots, N-1$
  Calcula a variável macroscópica: $v^{n+1}(\mathbf{x}) = \sum_{i=0}^{N-1} f_i(\mathbf{x}, t)$

  Propagação:
  para $i \leftarrow 1$ **até** $(N-1)/2$ **faça**
    $proximoX = X + e[i][0]$
    $proximoY = Y + e[i][1]$
    se $proximoX >= 0\ E\ proximoY >= 0\ E\ proximoX < Nx\ E\ proximoY < Ny$ **então**
      $fTmp = f[X][Y][i]$
      $f[X][Y][i] = f[X][Y][oposta(i)]$
      $f[X][Y][oposta(i)] = f[proximoX][proximoY][i]$
      $f[proximoX][proximoY][i] = fTmp$

5.3 Representação Esparsa

A simulação utilizando domínios regulares com o método de *lattice* Boltzmann é natural, mas no caso de simulações da atividade elétrica do coração surge a necessidade de representar sua complexa geometria. Para simular a eletrofisiologia cardíaca em geometrias irregulares foram implementadas duas estratégias: simulação da geometria irregular contida dentro de um domínio regular, usando o endereçamento direto, e a simulação utilizando um domínio irregular, através do endereçamento indireto. As abordagens de endereçamento direto e indireto serão apresentados a seguir.

A primeira estratégia consiste em considerar a malha do ventrículo dentro de um domínio regular, como mostra a Figura 5.4 e, então, simular todo este domínio regular armazenado em um vetor, tratando o que faz parte do tecido cardíaco e o que não faz. Esta estratégia é mais custosa, pois utiliza mais memória e realiza mais processamento. A segunda estratégia consiste em utilizar uma representação esparsa da geometria usada na simulação, assim, apenas pontos da região do ventrículo serão considerados.

O Algoritmo 4 apresenta a primeira abordagem, onde é usado um domínio como na Figura 5.4 e as funções de distribuição são acessadas como

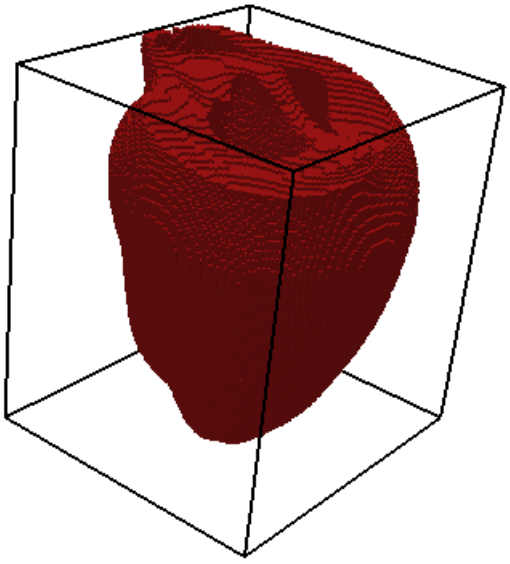


Figura 5.4: Domínio utilizado na implementação com matriz cheia.

`f[i*Nx*Ny*Nz+z*Ny*Nx+y*Nx+x]`. Para saber se um nó faz parte do domínio, foi utilizado um vetor auxiliar, chamado `tecido`, que é acessado da mesma forma que o vetor `f`. Cada posição do vetor `tecido` pode ter o valor 0 se o nó correspondente não é tecido e 1 se for tecido. A resolução das EDOs e as etapas de colisão e propagação só são executadas em um nó se ele pertencer à região do tecido. O tratamento das condições de contorno continua sendo feito só quando um nó pertencer ao contorno.

Para melhor representar geometrias irregulares, como aquela apresentada na Figura 5.4, uma estrutura de dados esparsa (Bernaschi *et al.*, 2010), ou endereçamento indireto, foi utilizada ao invés do esquema de endereçamento direto abordado anteriormente. A representação esparsa requer um vetor de adjacências de $N_{nos} \times N - 1$ inteiros para armazenar as informações sobre os vizinhos de um nó ao longo de cada direção $\mathbf{e}_i$. Como não há propagação na direção $\mathbf{e}_0$, não existem vizinhos para armazenar na lista de adjacência para esta direção.

A geometria irregular utilizada neste trabalho não possui um contorno suave, pois o domínio simulado é discretizado em pequenos cubos devido à natureza regular do método de *lattice* Boltzmann. Assim a superfície dos ventrículos não se aproxima tanto da geometria real. Uma forma de contornar este problema seria a utilização de uma condição

Algoritmo 4: Algoritmo para geometrias irregulares com endereçamento direto.

```
1  Configuração dos parâmetros do modelo
2  Inicialização do MLB
3  Inicialização das EDOs
4  para i ← 1 até numIteracoes faça
5      t ← t + Δt
6      para cada nó x faça
7          se tecido[x] == 1 então
8              Calcula as variáveis de estado η^n(x, t) usando o método Rush-Larsen
9      para cada nó x faça
10         se tecido[x] == 1 então
11             Colisão e propagação
12     para cada nó do contorno faça
13         Condição de contorno usando Eq. (3.30)
14     Grava dados em arquivo
```

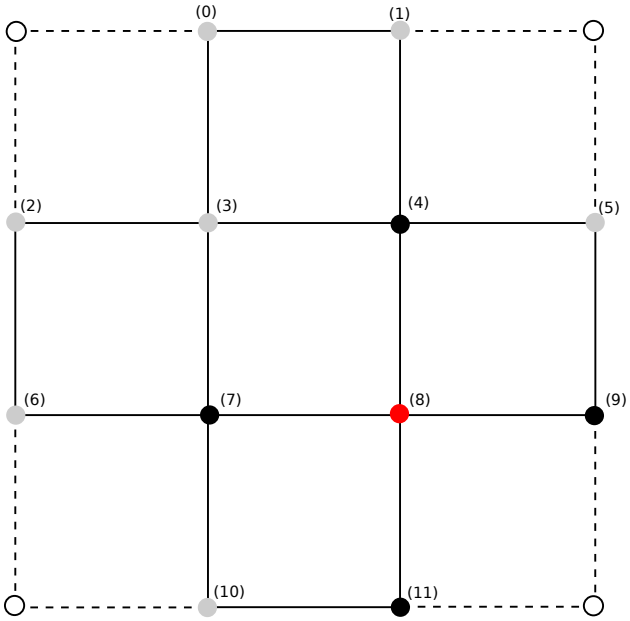
de contorno diferente, como a de Bouzidi *et al.* (2001) que consegue aproximar o contorno de forma suave.

A Figura 5.5 ilustra melhor a ideia dos endereçamentos direto e indireto para um *lattice* em duas dimensões, onde cada nó possui 4 vizinhos. No endereçamento direto, para um nó `k` com índices `ix`, `iy` e `iz`, pode-se acessar seu vizinho à esquerda simplesmente usando o índice `ix-1`, isto é, `tecido[ix-1][iy][iz]`. Se o nó `k` fosse o nó 0 então o nó à esquerda seria um nó que não pertence à geometria de interesse da simulação, portanto `tecido[ix-1][iy][iz] = 0`. O nó à direita de 0 seria o nó 1, que pertence à geometria do tecido cardíaco usada na simulação, consequentemente `tecido[ix+1][iy][iz] = 1` e as variáveis do modelo neste nó podem ser acessadas usando os mesmos índices da variável `tecido`.

Já no endereçamento indireto, primeiramente é necessário consultar a lista de adjacência `adj[k][0]` para encontrar o índice do vizinho à esquerda e então proceder com os cálculos. Neste exemplo foi considerado que o vizinho à esquerda de um nó possui índice da coluna igual a 0.

Como exemplo, considere o nó número 8 da Figura 5.5, em vermelho. Pode-se consultar a matriz de adjacência e descobrir cada um de seus vizinhos da seguinte forma: o vizinho à esquerda do nó 8 é $V[Adj[8][2]] = 7$, o vizinho à direita é $V[Adj[8][3]] = 9$, o vizinho superior é $V[Adj[8][0]] = 1$ e o inferior é $V[Adj[8][1]] = 11$. Em três dimensões, como foi utilizado o lattice D3Q7, a matriz de adjacência possui 6 colunas para representar

os vizinhos de cada nó. Quando o valor de uma posição da matriz de adjacência é -1 significa que não existe vizinho naquela direção e portanto o nó pertence ao contorno. Na



Matriz de Adjacência				
Nó	Norte (0)	Sul (1)	Esquerda (2)	Direita (3)
0	-1	3	-1	1
1	-1	4	0	-1
2	-1	6	-1	3
3	0	7	2	4
4	1	8	3	5
5	-1	9	4	-1
6	2	-1	-1	7
7	3	10	6	8
8	4	11	7	9
9	5	-1	8	-1
10	7	-1	-1	11
11	8	-1	10	-1

Figura 5.5: Representação esparsa.

etapa de colisão não é necessário a utilização da matriz de adjacência, pois esta etapa é puramente local e não depende de outros nós, já as etapas de propagação e tratamento das condições de contorno utilizam esta estrutura para fazer as operações no *lattice*.

Em um primeiro momento pode-se achar que a estrutura esparsa requer mais memória, devido ao uso de vetores auxiliares adicionais. Mas, para geometrias como a da Figura 6.9, a estrutura esparsa utilizou apenas 37% da memória necessária para o esquema de endereçamento direto com $\Delta x = 0.125$mm e $\Delta x = 0.25$mm, como pode ser visto na Figura 5.6.

5.4 Implementação paralela

A implementação paralela foi realizada usando a plataforma NVIDIA CUDA, com o objetivo de aumentar o desempenho em relação à implementação sequencial. A versão paralela do código também faz uso da estrutura de dados esparsa e do algoritmo *swap* apresentado. Para melhor desempenho e organização, o código foi dividido em quatro CUDA *kernels*:

- `estimulos`: que aplica estímulos às células, de acordo com sua localização e tempo

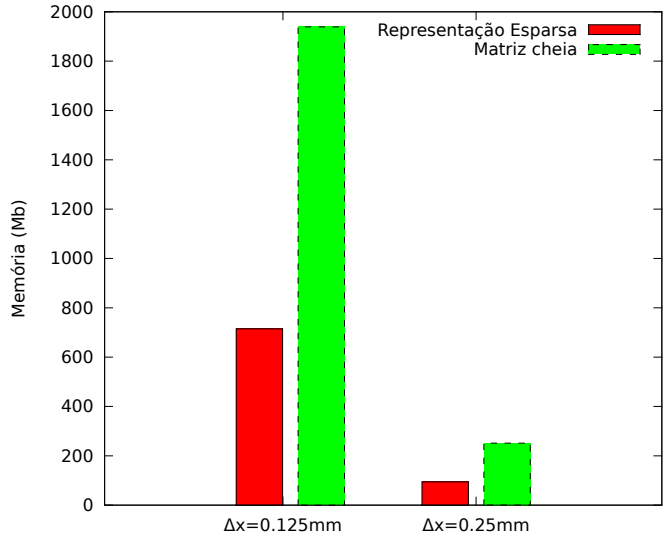


Figura 5.6: Comparação do uso de memória pelas implementações com representação esparsa e matriz cheia.

predefinido. Este *kernel* preenche um vetor com os estímulos para cada célula que será usado pelo *kernel* `resolveEDOs`.

- `inicializa`: para configurar a condição inicial do problema e iniciar as funções de distribuição das partículas.

- `colisaoPropagacaoContorno`: que realiza as etapas de colisão, propagação e contorno em todos os nós do *lattice*.

- `resolveEDOs`: para avançar as EDOs do modelo celular em todos os nós do *lattice* a cada passo tempo.

Na plataforma CUDA, os dados devem ser divididos em blocos e cada bloco é executado em um multiprocessador da GPU. Como os dados estão organizados em um vetor unidimensional, fica natural dividi-los também em blocos unidimensionais. Além disso, é necessário fazer um mapeamento do índice da *thread* para o índice correspondente no vetor, pois os dados estão organizados no formato de estrutura de *arrays* e a representação esparsa está sendo utilizada. Então as funções de distribuição das partículas são acessadas como `f[k + i*Np]`, onde `k` é o índice da *thread*, `i` é o índice da distribuição que será acessada e `Np` é o número de pontos do domínio.

O Algoritmo 5 apresenta como foi feita a implementação em CUDA, onde foram criados os quatro *kernels* apresentados. Primeiramente é feita a alocação das variáveis

na CPU e na GPU, em seguida é feita a leitura dos arquivos com informações sobre a malha, como coordenadas e conectividade dos nós. Então é chamado o *kernel* de inicialização e, em seguida, a cada passo de tempo são chamados os *kernels*: `estimulos`, `colisaoPropagacaoContorno` e `resolveEDOs`. Além disso, em alguns passos de tempo os resultados são gravados em um vetor, para posterior visualização. Quem grava os dados é a CPU, então é necessário transferir os dados da GPU para a CPU quando o *loop* no tempo termina.

Algoritmo 5: 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  Ler dados sobre a malha e copia para a GPU usando cudaMemcpy
5  Chama kernel de inicialização dos dados
6  enquanto tempo ≤ tempototal faça
7      Chama kernel estimulos()
8      Chama kernel resolveEDOs()
9      Chama kernel colisaoPropagacaoContorno()
10 Copia dados da GPU para CPU, com cudaMemcpy e salva em arquivo
```

Um exemplo de chamada à função *kernel* é apresentada no Algoritmo 6, 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*.

Algoritmo 6: Chamada ao *kernel* `colisaoPropagacaoContorno`.

```
1 int blockszx = 256
2 dim3 dimBlock(blockszx)
3 dim3 dimGrid((Np + dimBlock.x - 1) / dimBlock.x)
4 colisaoPropagacaoContorno<<<dimGrid, dimBlock>>>(parametros)
```

O Algoritmo 7 apresenta os detalhes para o *kernel* `colisaoPropagacaoContorno`. Na linha 1 é encontrado o índice da *thread* e então verifica-se se a *thread* deve executar o código, pois dependendo do número de blocos criados podem existir mais *threads* do que dados, portanto as *threads* que não estão associadas a dados não devem executar o código do *kernel*. Nas linhas 3-6 são acessados o potencial transmembrânico e as funções de distribuição do passo de tempo anterior. Em seguida, são calculados as distribuições de equilíbrio (linhas 7-9) e o operador de colisão MRT (linhas 10-12). As funções de distribuição são atualizadas nas linhas 16-18 e, para terminar a parte da colisão, o novo

potencial transmembrânico é calculado na linha 19.

Nas linhas 20-22 é realizado o acesso ao vetor de conectividade, que armazena os índices dos vizinhos de cada nó. Esta informação é usada durante a etapa de propagação (linhas 24-38) para mover as funções de distribuições de um nó para seus vizinhos. A propagação só é realizada em uma determinada direção se o nó possuir vizinhos nesta direção (linhas 24, 29 e 34). O tratamento da condição de contorno de fluxo nulo já é realizado automaticamente durante a etapa de propagação (Latt, 2007).

Algoritmo 7: *kernel* `colisaoPropagacaoContorno`.

```
1  k = blockIdx.x * blockDim.x + threadIdx.x
2  se (k < Np) então
3      f0l = f[k + 0*Np]
4      ⋮
5      f6l = f[k + 6*Np]
6      phil = phi[k]
7      feq0 = w0 * phil
8      ⋮
9      feq6 = w6 * phil
10     omega0 = −M^{-1}SM(f − f^{eq}) + wΔtR
11     ⋮
12     omega6 = −M^{-1}SM(f − f^{eq}) + wΔtR
13     f0l = f0l + omega0
14     ⋮
15     f6l = f6l + omega6
16     f[k + 0*Np] = f0l
17     ⋮
18     f[k + 6*Np] = f6l
19     phi[k] = f0l + f1l + f2l + f3l + f4l + f5l + f6l
20     v2=connec[6*k + 1]
21     v4=connec[6*k + 3]
22     v6=connec[6*k + 5]
23
24     se v2 != −1 então
25         fTmp = f[k + 2*Np]
26         f[k + 2*Np] = f[k + Np]
27         f[k + Np] = f[v2 + 2*Np]
28         f[v2 + 2*Np] = fTmp
29     se v4 != −1 então
30         fTmp = f[k + 4*Np]
31         f[k + 4*Np] = f[k + 3*Np]
32         f[k + 3*Np] = f[v4 + 4*Np]
33         f[v4 + 4*Np] = fTmp
34     se v6 != −1 então
35         fTmp = f[k + 6*Np]
36         f[k + 6*Np] = f[k + 5*Np]
37         f[k + 5*Np] = f[v6 + 6*Np]
38         f[v6 + 6*Np] = fTmp
```

6 Resultados

Neste capítulo são apresentados os experimentos numéricos realizados neste trabalho, além do ambiente computacional usado para executar os experimentos. Será apresentado também uma comparação entre os resultados obtidos com a implementação paralela usando GPU e a implementação sequencial.

6.1 Ambiente de desenvolvimento das simulações

Os experimentos numéricos foram realizados em uma máquina com Linux 2.6.32, com 12 GB de memória e dois processadores Intel Xeon E5620 2.4 GHz com 4 núcleos e 12 MB de memória cache cada. A implementação sequencial do MLB foi feita em C++ e compilada com o compilador GNU GCC versão 4.4.7, com as seguintes *flags* de otimização habilitadas: `-O3 -ffast-math`. Além disso, a máquina conta com uma GPU NVIDIA Tesla M2050, com 448 núcleos, distribuídos entre os 14 multiprocessadores e 6GB de memória, como apresentado na Tabela 4.1. O código CUDA foi compilado usando o compilador NVIDIA nvcc versão 6.0.1, com as *flags* de otimização `-O3 -use-fast-math`. As simulações foram executadas usando precisão simples e dupla para aritmética de ponto flutuante. As comparações entre código sequencial e paralelo sempre usaram a mesma representação de ponto flutuante. Todas medidas de tempo foram realizadas três vezes e calculou-se o tempo médio. O desvio padrão relativo máximo encontrado foi de 0.84%

6.2 Experimentos

Quatro problemas diferentes foram considerados com o objetivo de avaliar o código MLB para a simulação da atividade elétrica do tecido cardíaco. Alguns destes problemas usam geometrias simplificadas e outros usam geometrias realísticas dos ventrículos esquerdo e direito do coelho. O primeiro problema foi um *benchmark* para avaliar o MLB na resolução de problemas da eletrofisiologia cardíaca. Em seguida, para avaliar o desempenho da implementação paralela, foi feita uma simulação de uma região cúbica, submetida a diferentes refinamentos. Além disso, a configuração dos blocos na GPU foi variada, a fim

de melhorar o desempenho. Outro experimento foi a simulação da geometria biventricular do coelho, que foi dividida em duas partes: simulação da propagação da onda elétrica a partir de um estímulo no ápice do coração e a simulação de uma arritmia reentrante, ambas executadas nos ambientes sequencial e paralelo. Por último, foi realizada a verificação da convergência espacial do MLB para o problema da eletrofisiologia cardíaca usando uma geometria tridimensional regular.

6.2.1 Validação usando o Benchmark para eletrofisiologia

Um problema do tipo *benchmark* foi proposto por Niederer *et al.* (2011) para a validação de implementações do modelo monodomínio, uma vez que o modelo não possui solução analítica. O domínio consiste de uma região retangular de tamanho $20 \times 7 \times 3$ mm^3, que pode ser visto na Figura 6.1. As fibras são definidas como paralelas à direção longitudinal e o tensor de condutividade é considerado transversalmente isotrópico, como definido pela equação (2.20). O valor da condutividade na direção da fibra é de 0.1334 mS/mm, enquanto a condutividade na direção transversal à fibra é de 0.0176 mS/mm. Outros parâmetros são definidos como: $\chi = 140$ mm^{-1} e $C_m = 0.01$ μF/mm^2. O modelo celular utilizado foi o de ten Tusscher e Panfilov (2006). Com o objetivo de iniciar a atividade elétrica do coração, um estímulo externo de -50 μA/mm^3 foi aplicado durante 2 ms em uma pequena região cúbica S de 1.5 mm^3, representada na Figura 6.1.

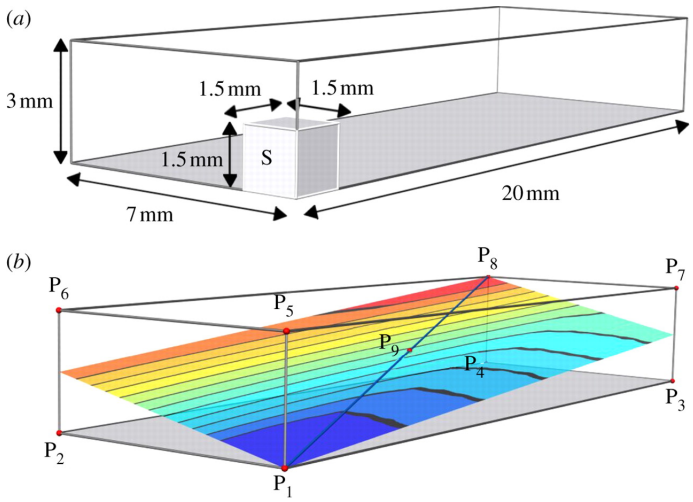
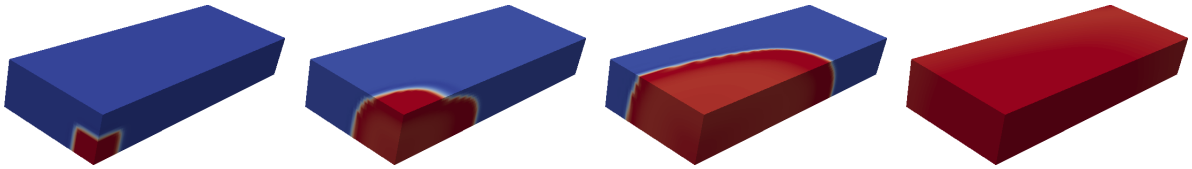


Figura 6.1: Esquema da simulação do *benchmark* (extraído de Niederer *et al.* (2011)). (a) Protocolo de estímulo, mostrando a região que foi estimulada. (b) Pontos do domínio onde foi avaliado o tempo de ativação.

O *benchmark* foi resolvido usando o método de *lattice* Boltzmann com diferentes passos

de tempo ($\Delta t = 0.05, 0.01$ e 0.005 ms) e espaçamentos ($\Delta x = 0.5, 0.2$ e 0.1 mm). O tempo de ativação, definido como o tempo em que o potencial transmembrânico v atinge o valor 0 mV, foi usado para medir a acurácia da implementação. Em Niederer *et al.* (2011) o tempo de ativação foi reportado em pontos ao longo da linha diagonal do domínio para várias implementações de diferentes grupos de pesquisa, utilizando diferentes métodos numéricos. A propagação da onda que ativa eletricamente o tecido cardíaco, da aplicação do estímulo elétrico até sua completa ativação, são mostradas nas Figuras 6.2a à d.



(a) (b) (c) (d)

Figura 6.2: Propagação da onda elétrica através do domínio definido pelo benchmark em diferentes passos de tempo.

Na Figura 6.3 são apresentados os tempos de ativação de dois códigos, CARP desenvolvido por Vigmond *et al.* (2003) e Chaste desenvolvido por Mirams *et al.* (2013), cujos resultados foram reportados em Niederer *et al.* (2011) e representam os diferentes comportamentos obtidos pelos códigos testados no experimento do *benchmark*. Os resultados do MLB para o *benchmark* também são mostrados na Figura 6.3 para o menor passo de tempo $\Delta t = 0.005$ ms. O tempo de ativação resultante da simulação com o MLB também converge para o mesmo valor obtido pelos outros códigos, que é aproximadamente 41 ms. Além disso, pode-se observar que o comportamento do MLB é muito similar ao apresentado pelo código Chaste.

Também foi realizado um estudo de convergência com relação ao passo de tempo Δt usado para as simulações, mantendo o espaçamento Δx constante. Para verificar a convergência com respeito ao passo de tempo, foram realizadas simulações com os seguintes passos de tempo $\Delta t = \{0.01, 0.005, 0.0025, 0.00125, 0.000625\}$ ms. Como no experimento anterior, o tempo de ativação do ponto P8 da Figura 6.1 foi usado como referência.

A Figura 6.4 mostra os resultados deste experimento. Nesse caso, o tempo de ativação não muda muito, mas para passos de tempo maiores a velocidade de condução da onda elétrica que se propaga pelo tecido é mais lenta e assim, o tempo de ativação no ponto de referência é maior. Porém, a figura mostra que ao reduzir o Δt, o valor do tempo de

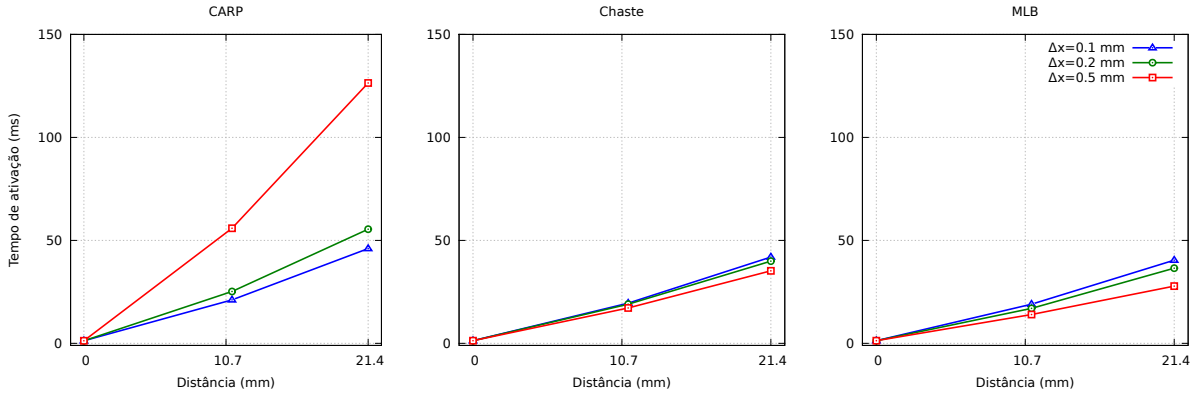


Figura 6.3: Tempo de ativação dos códigos de CARP e Chaste, como reportados em Niederer *et al.* (2011), e o tempo de ativação obtido com a presente implementação usando MLB com $\Delta t = 0.005$ ms.

ativação no ponto de referência converge para aproximadamente 40ms.

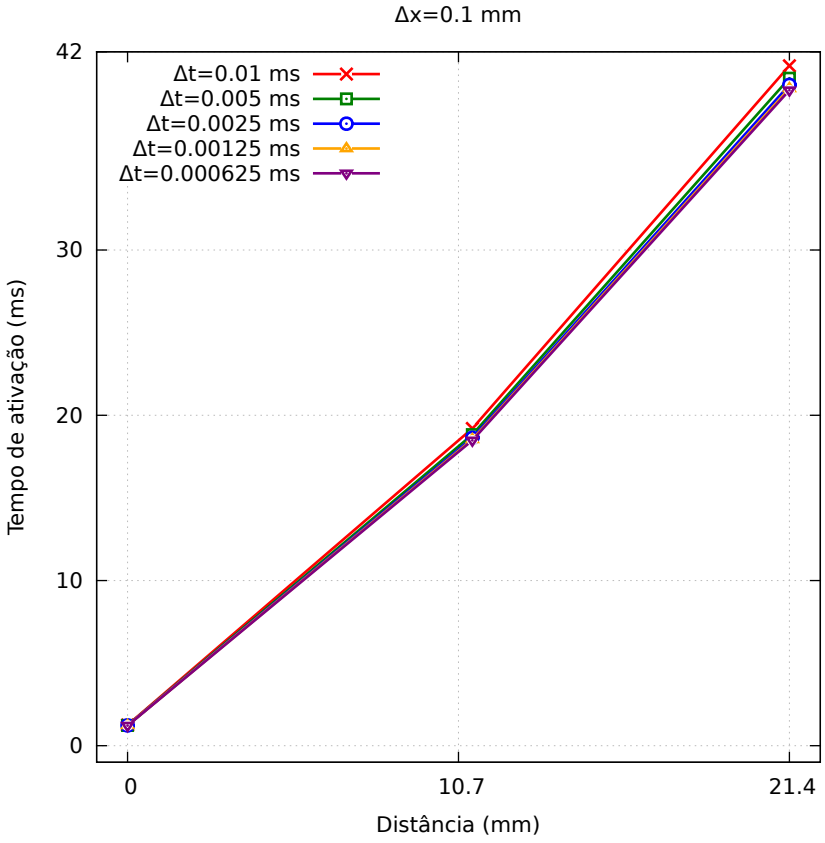


Figura 6.4: Tempos de ativação obtidos com a presente implementação usando o MLB, variando o passo de tempo Δt para um espaçamento fixo $\Delta x = 0.1$ mm. O gráfico mostra a convergência com respeito ao passo de tempo.

6.2.1.1 Comparação entre os códigos do *benchmark*, MVF e MLB

Uma análise mais aprofundada sobre a solução encontrada pelo MLB foi realizada, com o intuito de melhor validar o método para a aplicação proposta. Para isto usou-se o *benchmark* citado anteriormente, que possui os tempos de ativação para alguns pontos do domínio, encontrados por 11 códigos de diferentes grupos de pesquisas que simulam a eletrofisiologia cardíaca, dos quais 6 usam o MEF e os outros 5 usam o MDF. Através dos dados presentes no *benchmark* foi calculado o erro relativo entre os tempos de ativação de cada código no ponto P8, representado na Figura 6.1(b) . Para o cálculo do erro foi utilizada a instância mais refinada, com $\Delta x = 0.1mm$ e $\Delta t = 0.005ms$. O erro relativo foi calculado como

$$Erro = \frac{|MLB - Codigo|}{Codigo}. \tag{6.1}$$

Após o cálculo do erro para cada código, foi contabilizado quantos códigos quando comparados com o MLB tiveram erro menor do que 5% e erro menor que 10%. Assim, foi possível observar que o erro do MLB comparado com 3 códigos foi menor do que 5%. E o erro do MLB foi menor que 10% na comparação com 8 códigos, como pode ser observado na Tabela 6.1. O Apêndice A apresenta os tempo de ativação para todos os pontos observados no *benchmark* de cada um dos códigos, que foram extraídos do material suplementar de Niederer *et al.* (2011).

Tabela 6.1: Erro entre o tempo de ativação no ponto P8 quando MLB é comparado com outros códigos.

Código	Erro
Chaste	0.0371287
CARP	0.0895263
Richard Clayton	0.0481599
EMOS	0.131606
Alan Garny	0.0554554
acCELLerate/PETSc MultiDomain	0.0659831
Sander Land	0.14016
Alan Benson	0.0513923
OpenCMISS	0.140336
Elizabeth M. Cherry e Flavio H. Fenton	0.0670229
FEniCS/PyCC	0.0371287

Além desses códigos, o mesmo procedimento foi realizado para comparar o MLB à uma implementação do monodomínio usando o método dos volumes finitos (MVF) desenvolvida por Oliveira *et al.* (2012) e o erro encontrado comparando estas duas

implementações foi de 3.46%. Nesta simulação também foram utilizados $\Delta x = 0.01cm$ e $\Delta t = 0.005ms$.

A Figura 6.5 mostra os tempos de ativação nos pontos sobre a diagonal do domínio simulado, comparando o MLB com o MVF. Pode-se observar que o tempo de ativação nos dois casos converge para valores próximos de 41 ms no caso mais refinado, porém o comportamento de convergência dos métodos é diferente. No MLB o tempo de ativação é pequeno para discretizações grosseiras e ao refinar o tempo de ativação aumenta até convergir. Já na implementação do MVF, o tempo de ativação oscila entre valores altos e baixos até a convergência.

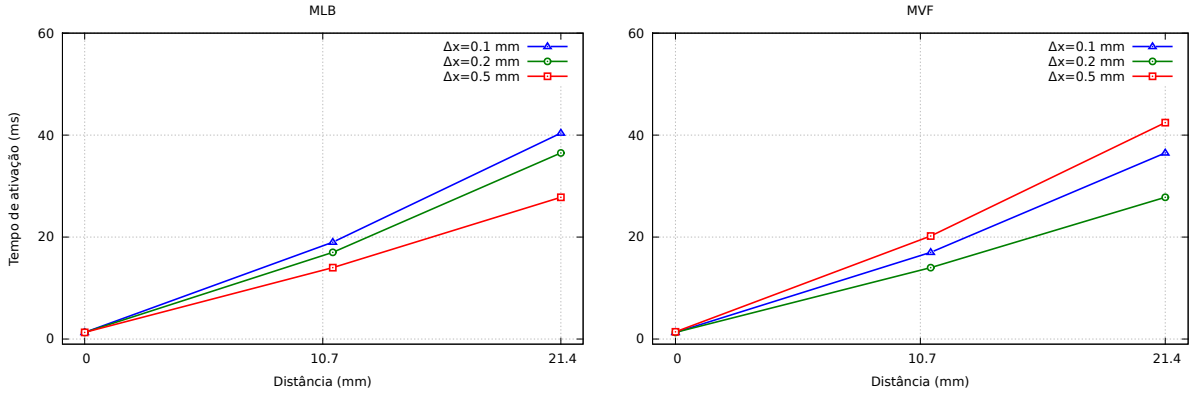


Figura 6.5: Tempo de ativação do MLB e MVF, usando $\Delta t = 0.005$ ms.

6.2.2 Convergência do MLB

Neste experimento foi verificada a ordem de convergência no espaço para o MLB na resolução do problema da eletrofisiologia cardíaca, isto é, a velocidade com que o erro da solução encontrada diminui à medida que a discretização espacial do domínio aumenta. O MLB é um método explícito de convergência quadrática no espaço, como verificado por Golbert (2009) para um problema da dinâmica dos fluidos.

Para verificar a convergência do MLB na resolução do modelo monodomínio foi considerada uma região cúbica de tamanho $1.28 \times 1.28 \times 1.28$ cm^3, usando um passo de tempo de $\Delta t = 0.01$ ms e espaçamentos $\Delta x = 0.08, 0.04, 0.02, 0.01$ cm. O modelo celular utilizado foi o de Mitchell e Schaeffer (2003), com um estímulo aplicado na metade do domínio nos 2 ms iniciais e a simulação foi executada até 5 ms. Então o potencial em todos os nós do domínio no último passo de tempo da simulação foi utilizado para o cálculo do erro em relação à solução referência. Como não se possuía a solução exata do

modelo, o resultado encontrado com o espaçamento de $\Delta x = 0.005$ cm foi utilizado como solução de referência. O erro entre a solução de referência e a solução de um determinado espaçamento foi calculado usando a versão discreta da norma L2

$$Erro = \sqrt{(\Delta x)^3 \sum_{i=0}^{L} (V(i) - V_{ref}(s \cdot i))} \tag{6.2}$$

onde L é o número de pontos do lado da região cúbica e s é um deslocamento utilizado para encontrar o ponto da malha de referência que corresponde ao ponto i da malha menos refinada.

A Figura 6.6 mostra a convergência do MLB neste experimento, onde pode-se notar a convergência quadrática a partir de $\Delta x = 0.04$ cm.

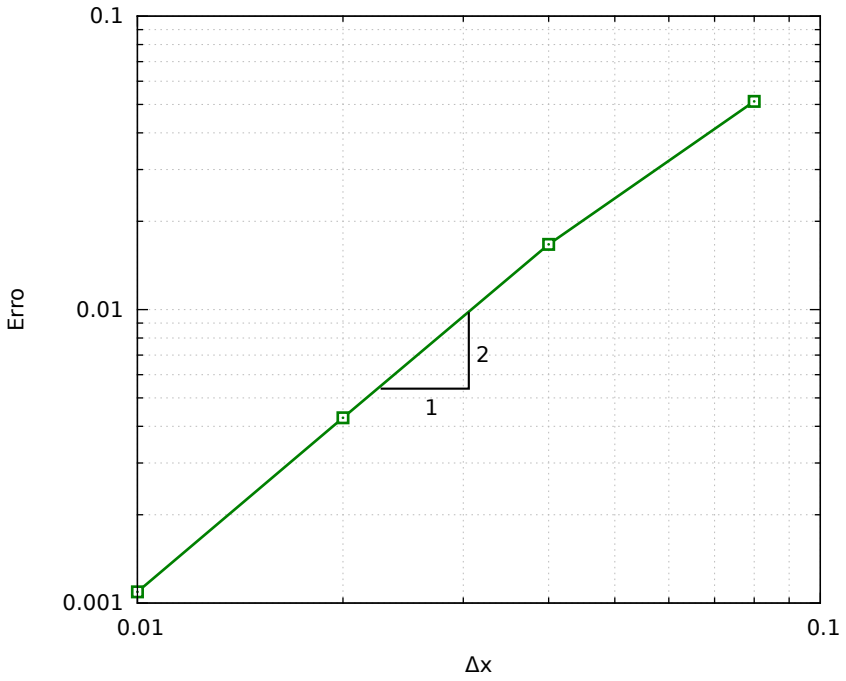


Figura 6.6: Convergência espacial do MLB na solução do modelo monodomínio acoplado ao modelo celular de Mitchell e Schaeffer (2003).

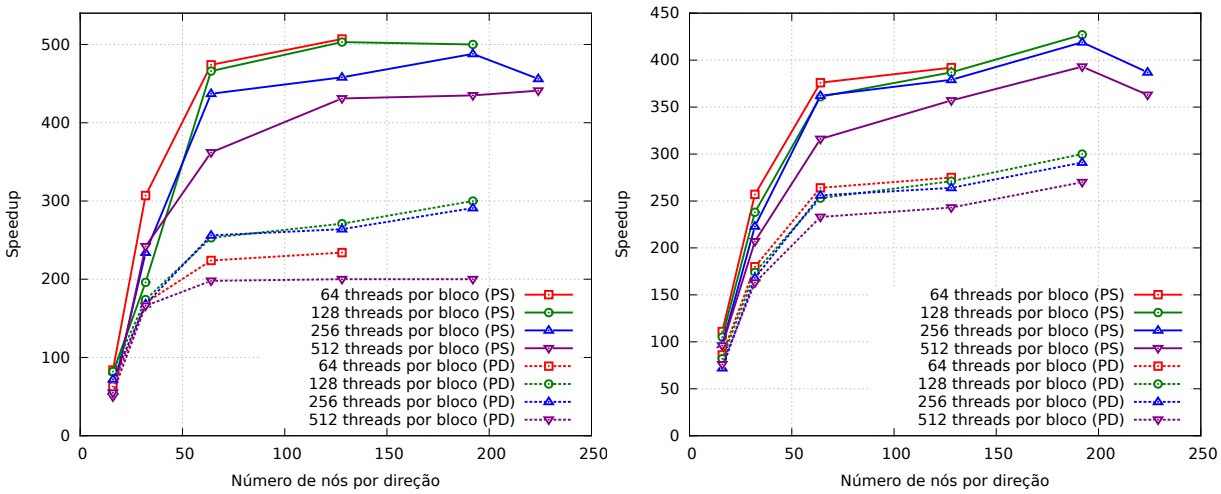
6.2.3 Atividade elétrica em um domínio regular

Para avaliar o desempenho da implementação paralela, uma versão modificada do *benchmark* foi usada. O domínio consiste de um cubo com dimensões 12.8 × 12.8 × 12.8 mm^3. Os parâmetros, condutividade e protocolo de estimulação foram os mesmos utilizados na simulação do *benchmark*. O desempenho foi avaliado

usando diferentes instâncias do problema, aumentando o número de nós das três dimensões do cubo, portanto a quantidade de pontos no domínio é dada por $N_p = 16^3, 32^3, 64^3, 128^3, 192^3, 224^3$. Não foi possível utilizar $N_p = 256^3$ pois não havia memória disponível, portanto foi utilizado a maior dimensão possível para o lado do cubo, que é $N_p = 224^3$. Foram usados dois modelos celulares com diferentes níveis de complexidade: LRI e TT2, para simular 1 s de atividade elétrica usando um passo de tempo $\Delta t = 0.1$ ms.

Os resultados de desempenho da implementação paralela do MLB usando GPU, para uma região cúbica do tecido cardíaco, são apresentados na Figura 6.7, que mostra a aceleração obtida. Os resultados reportados neste teste foram obtidos usando diferentes configurações de execução para os *kernels*, variando o número de *threads* por bloco, $n_{tb} = 64, 128, 256, 512$. A aceleração apresentada considera o tempo de resolução completa do problema, ou seja, inclui a resolução das EDOs e da EDP parabólica para todos os passos de tempo.

As acelerações obtidas para o modelo LRI foram maiores do que as obtidas para o modelo TT2, pois este último é mais complexo e então envolve mais operações de ponto flutuante e transações de memória. Em precisão simples, acelerações por volta de 500× foram obtidas para alguns casos usando o modelo LRI. Ao passo que acelerações por volta de 400× foram alcançadas com o modelo TT2. Em precisão dupla foi observado um bom desempenho também, mas as acelerações foram reduzidas à metade, como esperado.



(a) LRI. (b) TT2.

Figura 6.7: Desempenho do código para o *lattice* D3Q7 com diferentes discretizações do domínio cúbico, usando precisão simples (PS) e precisão dupla (PD).

A melhor configuração de blocos foi $n_{tb} = 64$ *threads*, representada pelas linhas vermelhas da Figura 6.7, indicando as maiores acelerações alcançadas neste experimento. Apesar do maior desempenho, esta configuração possui limitação com relação ao número de blocos que podem ser criados, onde o número máximo é de 65535, portanto para simular domínios com tamanho maior do que 65535 pontos seria necessário um número maior de *threads* por bloco. Analisando as configurações de blocos que conseguiram simular todos os tamanhos propostos, a melhor foi com $n_{tb} = 256$.

Na Tabela 6.2 são reportados os tempos de execução sequencial e paralelo e as respectivas acelerações para as simulações usando $n_{tb} = 256$ e precisão simples. Primeiramente, nota-se que os tempos de execução paralelos para malhas pequenas foram próximos à simulação em tempo real, levando 6 s e 37 s de tempo de execução para simular 1 s de atividade elétrica para os casos com tamanho de 32^3 e 64^3 pontos, usando o modelo celular mais complexo TT2. As simulações com o modelo LRI chegam mais perto ainda do tempo real, gastando 3 s e 13 s de tempo de execução para realizar a mesma simulação, o que mostra que a complexidade do modelo celular influencia bastante no tempo de execução da simulação.

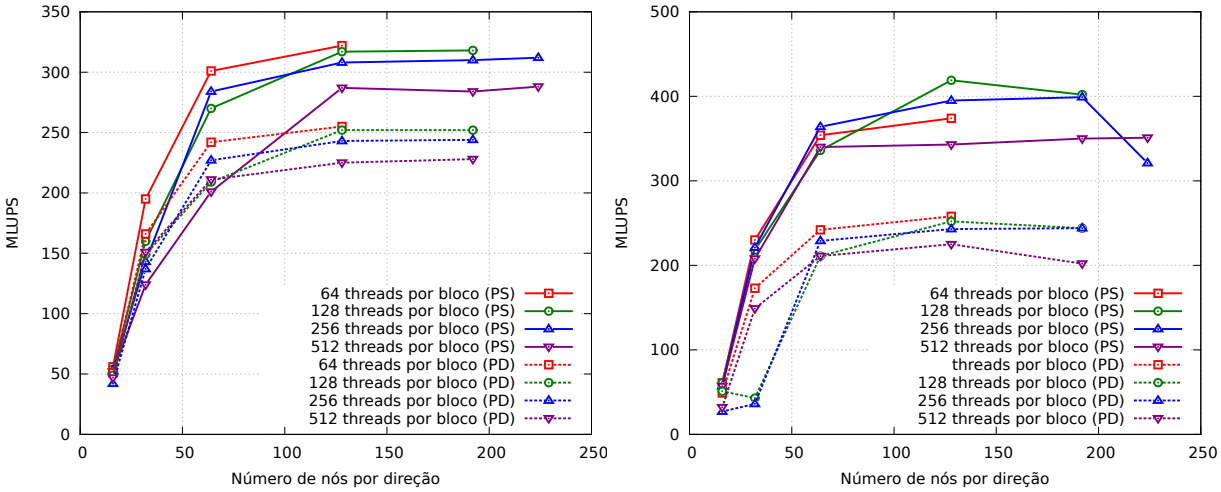
Diferentes acelerações foram obtidas para os dois modelos celulares, maiores acelerações para o modelo LRI (8 EDOs) e menores acelerações para o modelo TT2 (19 EDOs). Para uma malha com 192^3 pontos, foi alcançada uma aceleração de 488 em relação a versão sequencial, que demorava 1.7 dias para ser executada e passou a ser realizada em 5 minutos. Utilizando outros tamanhos de malha também foram alcançadas acelerações acima de 400×, nos casos com $N_p = 64^3, 128^3, 192^3, 224^3$ usando o modelo LRI, e $N_p = 192^3$ com o modelo TT2. A simulação mais demorada aconteceu com $N_p = 224^3$ e o modelo celular TT2, onde foram gastos 6.8 dias pela versão sequencial, enquanto a implementação em GPU gastou 25 minutos, obtendo uma aceleração de 387×.

Neste experimento a direção fibra está alinhada com o eixo x em todo o domínio, logo o tensor de condutividade resultante é diagonal. A implementação do termo não reativo da etapa de colisão dado pela equação (3.28), que utiliza este tensor de condutividade, necessita de uma multiplicação matriz-matriz. Com o objetivo de otimizar estas operações, foram consideradas apenas as operações da multiplicação que poderiam alterar o valor da mesma. Neste caso que o tensor é diagonal, foram consideradas duas implementações: uma que explora o fato do tensor ser diagonal e outra, mais geral, que considera o tensor

Tabela 6.2: Tempo de execução total e aceleração obtida para os modelos celulares LRI e TT2.

Modelo celular	Δx	N_{p}	CPU	GPU	*Speedup*
LRI	0.4 mm	32^3	689 s	3 s	230
	0.2 mm	64^3	5550 s	13 s	427
	0.1 mm	128^3	44400 s	97 s	458
	0.0667 mm	192^3	149000 s	305 s	488
	0.0571 mm	224^3	237000 s	519 s	457
TT2	0.4 mm	32^3	1420 s	6 s	237
	0.2 mm	64^3	13500 s	37 s	365
	0.1 mm	128^3	109000 s	287 s	380
	0.0667 mm	192^3	405000 s	966 s	419
	0.0571 mm	224^3	589000 s	1520 s	387

de condutividade com todas as componentes, sem explorar a estrutura diagonal.



(a) Tensor de condutividade completo. (b) Tensor de condutividade diagonal.

Figura 6.8: Desempenho do código para o lattice D3Q7 com diferentes discretizações para o domínio cúbico.

Os resultados de desempenho do MLB em termos de MLUPS nas Figuras 6.8a e b foram apresentados para as implementações genérica e diagonal, respectivamente. As simulações foram realizadas usando precisão simples (PS) e dupla (PD) para aritmética de ponto flutuante e com diferentes números de *threads* por bloco. Como o MLB é usado apenas para resolver a EDP parabólica e os resultados para os dois modelos celulares são muito próximos, então são apresentados apenas os resultados para o modelo TT2.

Primeiramente, é fácil de notar que a implementação diagonal tem um desempenho global melhor do que a implementação genérica, pois suas computações são simplificadas devido à estrutura diagonal do tensor de condutividade e assim menos operações de ponto

flutuante são necessárias. A implementação genérica obteve bons desempenhos, por volta de 320 MLUPS com 128 e 256 threads por bloco nas malhas mais refinadas com 224^3 nós. O melhor desempenho atingido pela implementação diagonal com a malha de 192^3 nós foi aproximadamente 400 MLUPS usando 256 *threads* por bloco em precisão simples, o que é compatível com outras implementações presentes na literatura (Bernaschi *et al.*, 2010).

Embora a implementação diagonal tenha obtido melhor desempenho, nota-se que este não é o caso geral e quando se tem geometrias complexas com orientações de fibras, representando dados fisiológicos como mostrado na Figura 6.9, é necessário usar uma implementação genérica. As duas implementações somente foram reportadas aqui para mostrar o impacto no desempenho que a anisotropia do tecido cardíaco provoca no operador de colisão MRT do MLB.

Na Seção 4.2.1.1 foi apresentado um modelo de desempenho, que mostra como calcular o número máximo de MLUPS que uma implementação do MLB pode alcançar. Fazendo os cálculos para a presente implementação em GPU, verifica-se que esta faz 37 operações de memória, entre leituras e escritas, portanto usando precisão simples $n_{bytes} = 148$ B. A largura de banda de memória da GPU usada é apresentada na Tabela 4.1, então o número máximo de MLUPS que pode ser alcançado com esta implementação é $MLUPS_{max} = 712$.

Analisando o número máximo de MLUPS que pode ser alcançado por este código, é possível verificar que no melhor resultado foi alcançado 58% de $MLUPS_{max}$, o que mostra um desempenho próximo ao alcançado em outros trabalhos (Habich *et al.*, 2011).

6.2.4 Geometria biventricular do coelho

Uma malha biventricular de coelho apresentada em Vetter e McCulloch (1998), com a direção de condutividade das fibras, foi usada para simulações anisotrópicas da atividade elétrica. Esta geometria do coelho foi obtida por um procedimento onde o coração do coelho foi mecanicamente fixado, os ventrículos foram preenchidos com silicone e então apropriadamente fatiado por imagens. A segmentação foi aplicada para cada imagem, com o objetivo de determinar o que era ou não tecido. Blocos de tecido foram extraídos destas fatias, e a orientação das fibras foi determinada. Então os dados foram ajustados para uma descrição de elementos finitos em coordenadas esféricas. Mais detalhes desses procedimentos podem ser encontrados em Vetter e McCulloch (1998). A versão desta malha usada no presente trabalho consiste de uma malha de hexaedros regulares,

adequados para as simulações do método de *lattice* Boltzmann.

Nestas simulações foram usadas duas malhas: a primeira com 522841 nós e um espaçamento de $\Delta x = 0.25$ mm; e a segunda é uma versão refinada com 3760560 nós e $\Delta x = 0.125$ mm. O modelo celular MSH para células ventriculares de coelhos foi usado para estas simulações, usando o método de Rush Larsen com um passo de tempo de $\Delta t = 0.1$ ms. Um tensor de condutividade transversalmente isotrópico foi usado com $\sigma_l = 0.1334$ mS/mm e $\sigma_t = 0.0176$ mS/mm. A geometria e a orientação das fibras são mostradas na Figura 6.9.

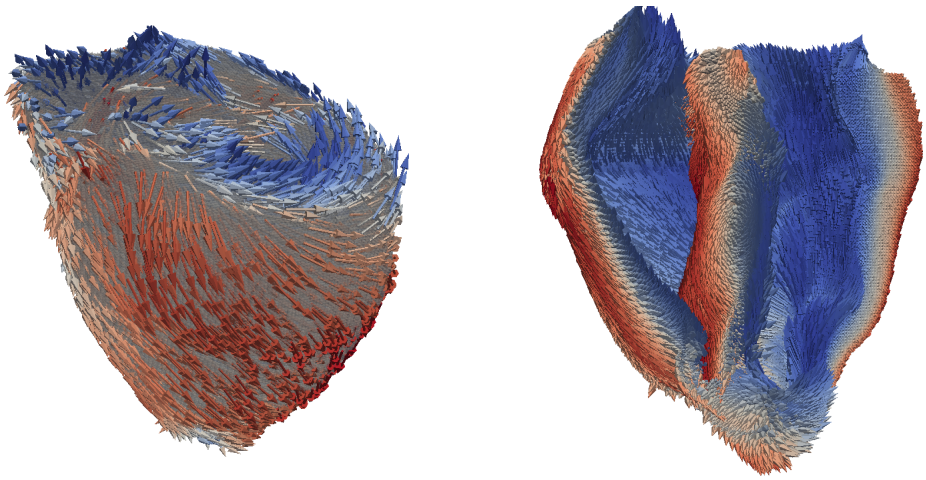


Figura 6.9: Geometria biventricular do coelho usada nas simulações e orientação das fibras. Os vetores que definem a orientação das fibras estão coloridos de acordo com a componente z.

A Figura 6.10 apresenta passos de tempo da simulação da atividade elétrica na geometria biventricular do coelho. Neste caso, foi aplicado um estímulo externo no ápice dos ventrículos.

Os resultados de desempenho para a geometria biventricular do coelho são mostrados na Tabela 6.3, onde o tempo de execução total, aceleração e MLUPS obtidos para o MLB completar 1 s de simulação usando o modelo celular MSH são detalhados.

Tabela 6.3: Tempo de execução e aceleração obtidos para a geometria do coelho.

Δx (mm)	T_{CPU}	T_{GPU}	*Speedup*	$\text{MLUPS}_{\text{GPU}}$
0.25	17700 s	85 s	208	265
0.125	134000 s	635 s	211	281

Para a malha mais refinada, uma aceleração de 211× foi alcançada pela implementação paralela usando GPU, em comparação com a versão sequencial usando um único núcleo de

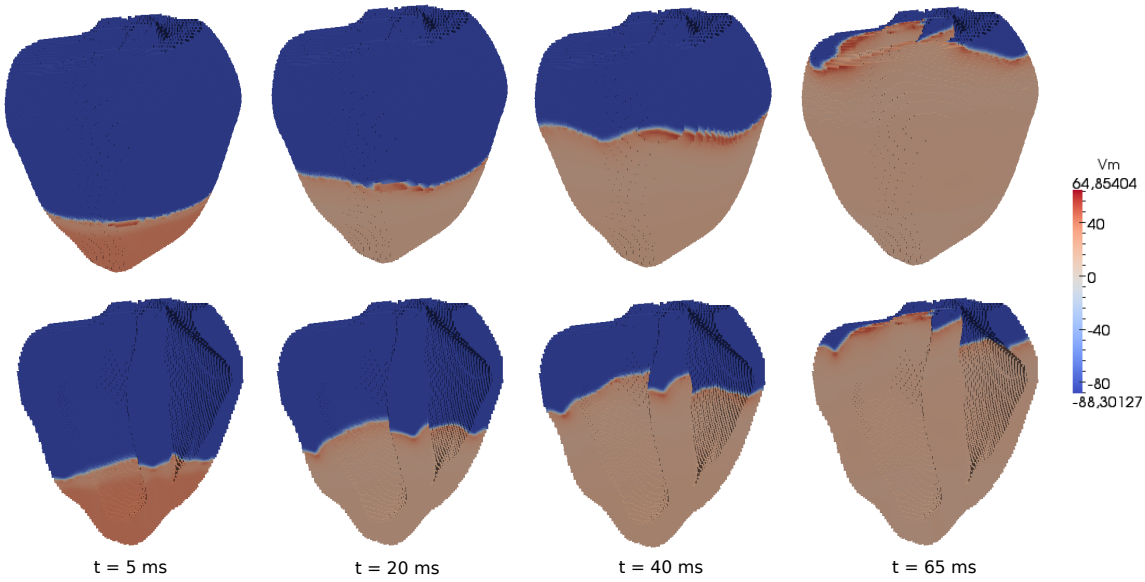


Figura 6.10: Simulação da atividade elétrica nos ventrículos do coelho.

um processador. Isto mostra claramente o bom ganho de desempenho quando comparado com a implementação sequencial. Neste teste 281 MLUPS foram obtidos com o MLB para a solução da EDP parabólica, e uma simulação que demorava em torno de 1 dia e meio, com a implementação sequencial, passa ser executada em aproximadamente 10 minutos com a implementação paralela utilizando GPU. Com a malha menos refinada ainda pode-se notar que a mesma simulação pode ser realizada em pouco mais de 1 minuto.

6.2.5 Arritmias cardíacas

Arritmia consiste em uma mudança desordenada no ritmo cardíaco, que pode fazer o coração bater mais rápido, mais devagar, ou bater de forma irregular. Quando o coração não bate em um ritmo apropriado, o bombeamento do sangue pode não ocorrer de forma efetiva, levando à danificação de órgãos, como o pulmão e o cérebro, além de poder ocasionar a morte. Um tipo de arritmia que provoca a desordem do ritmo cardíaco é a reentrante, onde a onda elétrica de excitação do coração forma uma espiral, provocando uma desorganização da excitação do tecido e consequentemente impedindo o bombeamento do sangue.

A desorganização do ritmo cardíaco pode ser causado pela propagação anormal da onda de excitação, que pode acontecer quando uma parte do tecido está impedida de sofrer excitação. Este período em que algumas células não podem ser excitadas é chamado

de período refratário absoluto, que ocorre no intervalo de tempo entre a despolarização e a repolarização das células. Após este período existe um pequeno intervalo onde as células são inibidas a um estímulo, mas ainda existe a possibilidade de excitação, que é o chamado período refratário relativo. Quando uma onda de excitação encontra células no período refratário absoluto, a propagação elétrica não acontece nesta direção e então a onda vai se propagar por onde houver células fora do período refratário absoluto. Consequentemente uma propagação anormal da onda de excitação se forma, podendo resultar em uma arritmia reentrante.

A Figura 6.11 ilustra um caso de arritmia reentrante em uma parte do tecido cardíaco. Inicialmente todas células podem ser excitadas, pois estão no potencial de repouso, em seguida, um estímulo é aplicado em uma parte do tecido como mostra a Figura 6.11(a). Assim uma onda de excitação é formada e se propaga na direção horizontal, então algumas células estarão no período refratário absoluto e outras fora deste período. Após um intervalo de tempo em que a onda está se propagando um segundo estímulo é aplicado em uma parte do tecido, como apresentado na Figura 6.11(b). Neste momento, a nova onda elétrica gerada não consegue se propagar na direção horizontal, pois as células à direita do estímulo estão no período refratário absoluto. Mas as células acima do estímulo estão fora deste período e já podem sofrer excitação novamente, portanto a onda gerada pelo segundo estímulo se propaga na direção vertical. Após um intervalo de tempo as células que estavam no período refratário absoluto deixam este período, então estas já podem ser excitadas novamente e uma espiral começa a ser gerada, como na Figura 6.11(c). A espiral pode se sustentar, como mostra a Figura 6.11(d), caracterizando uma arritmia reentrante que pode levar o paciente à morte. Este mecanismo de simulação de uma arritmia utilizando dois estímulos é chamado de protocolo S1-S2 e foi usado neste trabalho para a simulação de uma arritmia reentrante nos ventrículos de um coelho.

6.2.5.1 Simulação de arritmia

Utilizando a geometria biventricular do coelho, apresentada anteriormente, foi realizada uma simulação de arritmia reentrante. As configurações do modelo e do MLB foram as mesmas da simulação anterior, usando a geometria do coelho menos refinada. Apenas o parâmetro χ do monodomínio foi modificado de $\chi = 140$ mm^{-1} para $\chi = 600$ mm^{-1} e o tensor de condutividade considerado isotrópico, com o objetivo de facilitar a geração da

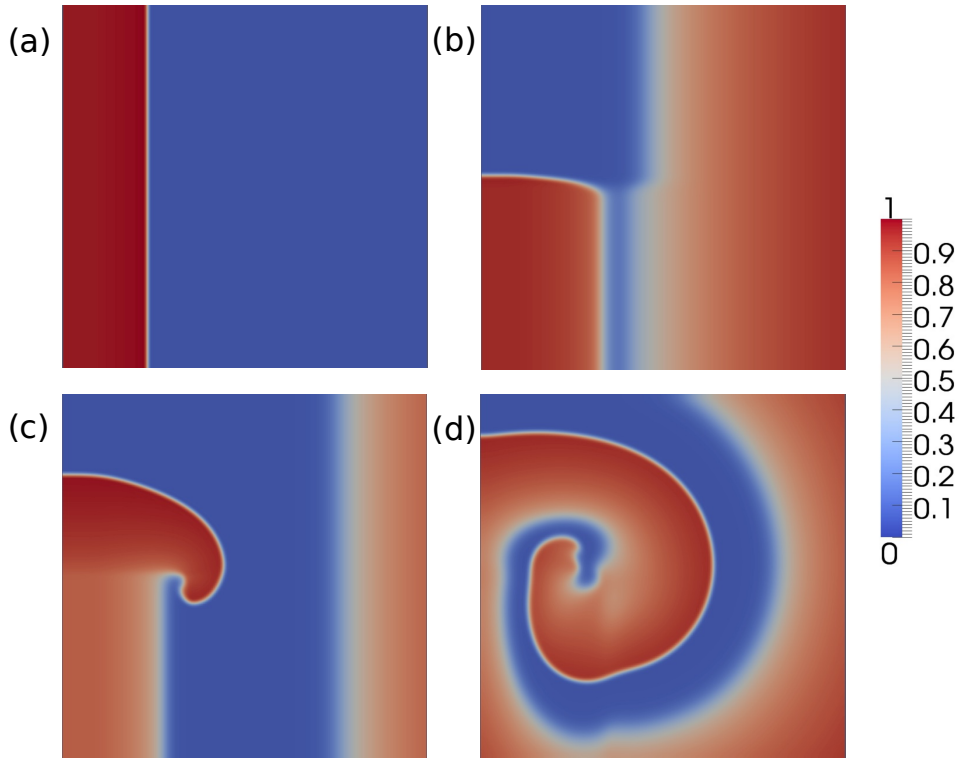


Figura 6.11: Exemplo da distribuição espacial do potencial transmembrânico na simulação do modelo monodomínio com o modelo celular de Mitchell e Schaeffer (2003) usando o método de *lattice* Boltzmann e Euler explícito durante 1 s (Campos, 2013) (a) Primeiro estímulo aplicado (b) Segundo estímulo aplicado (c) Formação de uma espiral (d) Espiral que mostra comportamento arrítmico do coração.

onda reentrante.

Para reproduzir tal fenômeno, uma estratégia semelhante com a apresentada na Seção 6.2.5 foi realizada, onde aqui é aplicado um primeiro estímulo no ápice dos ventrículos nos dois milissegundos iniciais, então uma onda elétrica se forma e propaga-se pelo tecido. Depois de um tempo, um segundo estímulo é aplicado em uma parte do coração ($t = 215$ ms), então uma segunda onda elétrica se forma propagando-se por onde existem células fora do período refratário, formando uma espiral reentrante que caracteriza um comportamento de arritmia.

O fenômeno foi simulado por 2 segundos e as duas malhas da simulação anterior foram utilizadas. Os tempos de execução e o desempenho alcançado com a implementação paralela podem ser vistos na Tabela 6.4. No caso mais refinado o MLB alcançou 228 MLUPS na resolução da EDP parabólica e a implementação paralela obteve um *speedup* de 197× em relação à implementação sequencial. Utilizando a malha menos refinada a simulação, que demorava por volta de 9 horas e meia, passa a ser executada em pouco mais de 3 minutos. Além disso, em todos os experimentos a espiral reentrante se manteve

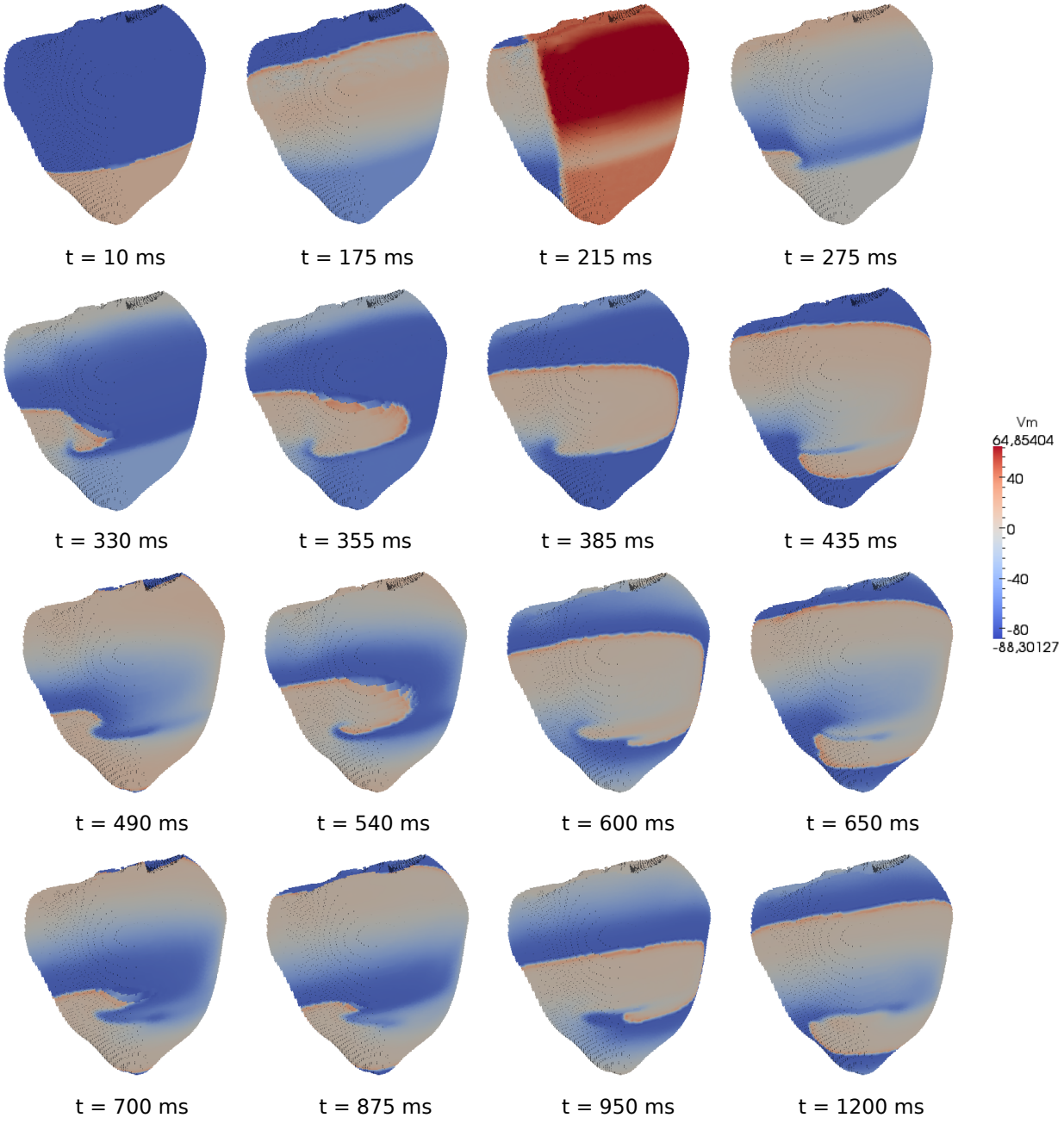


Figura 6.12: Simulação de arritmia cardíaca reentrante em diferentes passos de tempo.

por toda a simulação sem que um novo estímulo fosse aplicado.

Tabela 6.4: Tempo de execução e aceleração obtidos para a simulação de arritmia.

Δx (mm)	T_{CPU}	T_{GPU}	*Speedup*	$\text{MLUPS}_{\text{GPU}}$
0.25	34550 s	190 s	181	193
0.125	264000 s	1340 s	197	228

7 Conclusão

Neste trabalho foi apresentado o uso do método de *lattice* Boltzmann para a solução numérica de equações da eletrofisiologia cardíaca em domínios tridimensionais com geometrias regulares e também geometrias complexas do coração. Foi discutida uma implementação otimizada do MLB com melhoramentos na etapa de propagação usando o algoritmo *swap*, que exige menos memória e reduz o acesso a memória. Uma representação esparsa da malha do MLB também foi introduzida com o objetivo de reduzir o uso de memória.

Uma das principais características do MLB é a alta localidade de suas operações, assim sendo bastante adequado para implementações paralelas usando GPUs modernas. Uma implementação paralela da equação do monodomínio com o MLB foi apresentada usando o ambiente CUDA. Então, os ganhos de desempenho obtidos com esta estratégia foram mostrados através dos vários experimentos numéricos realizados. Foram obtidas acelerações de até 500×, o MLB alcançou 419 MLUPS e um desempenho próximo ao tempo real foi alcançado utilizando um computador equipado com uma GPU moderna, que torna esta proposta bastante adequada para aplicações práticas no contexto clínico. As simulações foram realizadas usando precisão simples e duplas para as operações de aritmética de ponto flutuante e com diferentes números de *threads* por bloco.

7.1 Trabalhos relacionados

Discute-se aqui alguns trabalhos com foco em desempenho de simulações da eletrofisiologia cardíaca usando aceleradores e também o método de *lattice* Boltzmann.

Bartocci *et al.* (2011) apresentou simulações bidimensionais da dinâmica cardíaca usando o método das diferenças finitas implementado em CUDA para resolver o modelo monodomínio. Cinco diferentes modelos celulares com diferentes níveis da complexidade foram usados para as simulações em malhas com tamanho variável para testar sua implementação. Os resultados mostraram um desempenho próximo ao tempo real. Neste trabalho, simulações com desempenho próximo ao tempo real foram apresentadas, além de ter sido apresentado um método diferente para a solução numérica do modelo

monodomínio em simulações tridimensionais com geometrias regulares e irregulares. Para fazer uma comparação com os resultados de Bartocci *et al.* (2011) em uma malha bidimensional de 1536^2 nós, foi considerado o mesmo modelo (TT2) e o mesmo tempo de simulação de 1 s. Nota-se que a GPU utilizada no presente trabalho é similar à usada por Bartocci *et al.* (2011), que foi uma NVIDIA Tesla C2070. Neste caso, sua melhor implementação que aproveita a sua malha bidimensional usando memória de textura e também outras otimizações para modelos celulares, levou aproximadamente 300 s para simular 1 s da dinâmica cardíaca em precisão simples. Nosso experimento com um domínio regular cúbico com 128^3 nós, ligeiramente menor do que sua malha, levou 287 s para completar a simulação e uma aceleração de 380× foi obtida. Deve-se observar que nossa implementação é genérica e usa uma estrutura de dados com endereçamento indireto que suporta domínios tridimensionais complexos. Deve ser lembrado que a implementação de Bartocci aproveita alguns recursos CUDA otimizados para localidade espacial em simulações bidimensionais.

Recentemente Wang *et al.* (2014) apresentou os resultados de desempenho da solução do modelo monodomínio em aceleradores modernos usando diferentes implementações com OpenMP, CUDA, OpenCL e também uma implementação usando OpenACC, uma interface de paralelização automática. Os resultados de desempenho foram apresentados em termos de aceleração e somente para simulações bidimensionais. O modelo celular usado em Wang *et al.* (2014) foi um modelo de oito variáveis, que em termos de complexidade é comparável ao modelo LRI usado em nosso trabalho. Em seu melhor cenário com uma malha bidimensional de 2048^2 nós usando precisão dupla, uma aceleração de 200× foi obtida em comparação com uma implementação sequencial executada em um único núcleo e 30× em comparação com um código paralelo usando OpenMP. No presente trabalho, em termos de aceleração somente, o melhor caso usando precisão dupla foi de 300× mais rápido do que a implementação sequencial executando em um único núcleo. Note que o melhor resultado de Wang *et al.* (2014) foi obtido com uma GPU Tesla K20, cuja capacidade de processamento é muito maior do que a da GPU usada neste trabalho.

Em termos de trabalhos relacionados usando o método de *lattice* Boltzmann para simulações da eletrofisiologia cardíaca usando o modelo monodomínio, pelo nosso conhecimento, Rapaka *et al.* (2012) foi o único trabalho a reportar seu uso em uma implementação sequencial usando um único núcleo. Neste trabalho foi usado o mesmo

modelo de *lattice* e o mesmo operador de colisão MRT, além de descrever uma implementação otimizada do MLB usando o algoritmo de *swap* e uma representação esparsa para a malha computacional (Campos *et al.*, 2015). Também é fornecida uma detalhada descrição de uma implementação paralela em GPU e seu desempenho para um problema de reação-difusão com o objetivo de explorar a alta localidade de suas operações para obter simulações rápidas e próximas do tempo real, que foi demonstrado através dos resultados.

É importante relacionar o presente trabalho com outros trabalhos de nosso grupo, tais como Oliveira *et al.* (2012) e Rocha *et al.* (2011) que têm foco em simulações rápidas da eletrofisiologia cardíaca. Em Rocha *et al.* (2011) simulações bidimensionais da atividade elétrica foram executas em GPU com CUDA usando o método dos elementos finitos para discretizações espaciais e o esquema Crank-Nicolson implícito no tempo. Assim, um sistema de equações lineares tem que ser resolvido em cada passo de tempo, ao contrário do MLB usado neste trabalho que é explícito. Em Oliveira *et al.* (2012) uma implementação paralela através do OpenMP com um algoritmo para malha adaptativa usando o método dos volumes finitos foi proposto para soluções numéricas das equações. Devido à estrutura de dados utilizada para refinamento da malha adaptativa, uma implementação eficiente em ambiente GPU ainda é um desafio. Apesar disso ressaltamos outras estratégias como aquela proposta por Neumann e Neckel (2013) para malha adaptativa com MLB que poderia ser combinada com o presente trabalho para simulações em GPU.

7.2 Trabalhos Futuros

Como já foi apresentado, o modelo monodomínio usado para reproduzir a atividade elétrica no tecido cardíaco é simplificado, com hipóteses que não são válidas, como taxas de anisotropias iguais entre os meios intra- e extracelular. Para melhorar o trabalho neste aspecto seria interessante usar o modelo Bidomínio (Amorim e dos Santos, 2013), que considera os meios intra- e extracelular e reproduz o fenômeno de forma mais realística. Usando este modelo seria possível reproduzir um eletrocardiograma ou ainda realizar uma simulação de desfibrilação. Apesar deste modelo ser mais robusto ele é composto por duas EDPs, uma parabólica e outra elíptica, cuja solução numérica possui um custo computacional maior em relação ao modelo monodomínio. Além disso, o custo

computacional para resolver equações elípticas no MLB é alto.

Outro aspecto que pode ser melhorado é o tratamento do contorno, de maneira a considerar geometrias realísticas do coração com contorno suave. Neste sentido, a condição de contorno de Bouzidi *et al.* (2001) poderia ser utilizada, que é uma combinação da condição de contorno *bounce-back* e uma interpolação espacial de primeira ou segunda ordem.

Com relação ao desempenho, observou-se que grande parte do tempo gasto para realizar a simulação é usado para resolver as EDOs dos modelos celulares. Otimizações podem ser feitas no modelo celular para reduzir o tempo gasto com sua solução, como por exemplo a estratégia de tabela *look up* (Campos, 2008).

No presente trabalho a implementação paralela usou apenas uma GPU e o processador só foi usado para cópia de dados e chamada dos *kernels*. Uma abordagem usando múltiplas GPUs juntamente com um balanceamento de carga dividindo os cálculos também com o processador, pode deixar a simulação ainda mais próxima do tempo real. Além disso, pode-se explorar ainda mais a implementação CUDA, estudando por exemplo a utilização da memória de textura que pode melhorar o desempenho.

APÊNDICE A - Benchmark

As seguintes tabelas apresentam os tempo de ativação para a simulação do *benchmark* em diferentes pontos. Cada tabela representa os resultados de uma implementação. A Tabela A.1 mostra os resultados obtidos com a implementação do presente trabalho, a Tabela A.13 apresenta os resultados para o código de Oliveira (2013) e as demais apresentam os resultados dos diferentes códigos testados em Niederer *et al.* (2011).

Tabela A.1: Tempos de ativação para a implementação do MLB realizada neste trabalho.

Δx	Δt	P1	P2	P3	P4	P5	P6	P7	P8	P9
0.1	0.005	1.225	31.45	7.05	32.525	22.6	39.45	24.225	40.4	18.825
0.1	0.01	1.25	32.6	7.25	33.6	23.35	40.25	24.9	41.15	19.2
0.1	0.05	1.5	39.25	8.75	40	28.25	46	29.5	46.5	21.75
0.2	0.005	1.225	28.625	5.525	29.6	18.7	35.45	20	36.25	17.075
0.2	0.01	1.25	29.7	5.65	30.55	19.2	36.2	20.55	36.95	17.4
0.2	0.05	1.5	36	6.75	36.75	23	41.25	24.5	41.75	19.75
0.5	0.005	1.25	23.275	3.675	23.95	13.05	27.125	13.4	27.75	13.825
0.5	0.01	1.25	23.9	3.75	24.7	13.3	27.75	13.75	28.35	14.15
0.5	0.05	1.5	29	4.5	29.5	15.75	32.75	16.5	33	16.25

Tabela A.2: Tempos de ativação para o código CARP.

Δx	Δt	P1	P2	P3	P4	P5	P6	P7	P8	P9
0.1	0.005	1.25	32.76	9.69	34.47	30.54	45.35	31.47	46.02	21.1
0.1	0.01	1.26	32.92	9.74	34.63	30.64	45.54	31.58	46.22	21.2
0.1	0.05	1.35	34.1	10.02	35.83	31.37	46.99	32.35	47.71	21.94
0.2	0.005	1.25	35.24	12.06	37.85	40.55	54.96	41.57	55.43	25.19
0.2	0.01	1.26	35.37	12.09	37.98	40.58	55.11	41.61	55.58	25.27
0.2	0.05	1.35	36.34	12.23	38.95	40.8	56.24	41.85	56.73	25.88
0.5	0.005	1.25	50.69	37.83	61.13	113.47	125.94	114.62	126.4	55.88
0.5	0.01	1.26	50.71	37.85	61.22	113.46	126.02	114.61	126.48	55.94
0.5	0.05	1.35	50.84	37.74	61.76	113.14	126.5	114.38	126.98	56.24

Tabela A.3: Tempos de ativação para o código Chaste.

Δx	Δt	P1	P2	P3	P4	P5	P6	P7	P8	P9
0.1	0.005	1.3	32.3	8.5	33.3	27.8	41.3	28.5	41.9	19.5
0.1	0.01	1.3	32.3	8.5	33.3	27.8	41.3	28.5	41.9	19.5
0.1	0.05	1.3	32.1	8.5	33.2	27.8	41.4	28.6	42.0	19.5
0.2	0.005	1.3	33.4	9.0	33.8	28.9	40.4	28.6	39.9	19.0
0.2	0.01	1.3	33.2	9.0	33.7	28.9	40.5	28.6	40.1	19.1
0.2	0.05	1.3	33.2	9.0	33.7	28.9	40.5	28.6	40.1	19.1
0.5	0.005	1.3	37.1	7.8	35.7	27.2	37.2	24.2	35.2	17.2
0.5	0.01	1.3	37.1	7.8	35.7	27.2	37.2	24.2	35.2	17.2
0.5	0.05	1.3	37.0	7.8	35.7	27.2	37.3	24.2	35.3	17.2

Tabela A.4: Tempos de ativação para o código de Richard Clayton.

Δx	Δt	P1	P2	P3	P4	P5	P6	P7	P8	P9
0.1	0.005	0.83	31.87	8.78	33.02	30.4	43.17	31.76	44.02	19.96
0.1	0.01	0.84	31.42	8.71	32.54	30.16	42.63	31.49	43.45	19.69
0.1	0.05	0.85	17.5	6.0	17.45	13.1	21.25	13.15	21.2	19.98
0.2	0.005	0.83	34.37	11.99	35.79	41.68	53.95	43.34	54.84	24.97
0.2	0.01	0.84	34.03	11.94	35.42	41.5	53.61	43.14	54.48	24.79
0.2	0.05	0.85	31.35	11.6	32.6	40.15	51.1	41.7	51.9	23.45
0.5	0.005	0.83	46.11	27.73	50.64	112.32	122.63	113.3	123.26	59.74
0.5	0.01	0.84	45.89	27.69	50.44	112.16	122.4	113.14	123.03	59.61
0.5	0.05	0.85	44.35	27.4	48.95	111.05	120.8	112.0	121.45	58.73

Tabela A.5: Tempos de ativação para o código EMOS.

Δx	Δt	P1	P2	P3	P4	P5	P6	P7	P8	P9
0.1	0.005	0.0	31.75	8.0	34.25	29.75	46.75	32.25	48.25	21.25
0.1	0.01	0.0	32.25	8.25	34.5	30.0	47.0	32.5	48.5	21.5
0.1	0.05	0.0	34.75	8.5	36.75	31.25	49.5	33.75	51.0	22.5
0.2	0.005	0.0	34.5	9.75	39.0	39.5	57.75	43.0	60.75	27.0
0.2	0.01	0.0	34.75	9.75	39.25	39.5	58.0	43.25	61.0	27.13
0.2	0.05	0.0	36.25	9.75	40.75	40.0	60.0	43.75	62.75	27.88
0.5	0.005	0.0	55.25	12.25	59.25	116.0	132.25	118.25	133.25	54.0
0.5	0.01	0.0	55.25	12.25	59.5	116.0	132.5	118.25	133.5	54.0
0.5	0.05	0.0	55.5	12.5	60.0	116.5	134.0	119.0	135.0	54.75

Tabela A.6: Tempos de ativação para o código de Alan Garny.

Δx	Δt	P1	P2	P3	P4	P5	P6	P7	P8	P9
0.1	0.005	1.22	32.2	9.25	33.44	30.4	43.42	31.86	44.36	20.07
0.1	0.01	1.22	32.0	9.23	33.22	30.31	43.19	31.76	44.11	19.95
0.2	0.005	1.22	34.57	12.45	36.14	40.76	53.48	42.7	54.53	24.12
0.2	0.01	1.22	34.43	12.43	36.0	40.7	53.39	42.64	54.44	24.06
0.2	0.05	1.19	33.34	12.29	34.82	40.26	52.7	42.17	53.72	23.64
0.5	0.005	1.22	45.1	32.06	53.21	118.27	129.35	120.37	130.74	55.11
0.5	0.01	1.22	45.03	32.03	53.16	118.19	129.26	120.29	130.66	55.06
0.5	0.05	1.19	44.53	31.8	52.74	117.55	128.62	119.65	130.02	54.71

Tabela A.7: Tempos de ativação para o código acCELLerate/PETSc MultiDomain.

Δx	Δt	P1	P2	P3	P4	P5	P6	P7	P8	P9
0.1	0.005	1.23	32.73	9.17	33.92	30.59	43.96	32.02	44.86	20.68
0.1	0.01	1.23	33.09	9.23	34.29	30.83	44.44	32.28	45.36	20.89
0.2	0.005	1.23	34.55	10.52	35.74	39.06	52.08	40.61	52.9	23.68
0.2	0.01	1.23	34.87	10.56	36.08	39.2	52.52	40.78	53.35	23.88
0.2	0.05	1.3	37.4	10.85	38.75	40.3	55.9	42.1	56.85	25.5
0.5	0.005	1.23	45.46	30.59	51.65	119.43	129.59	120.14	130.06	64.52
0.5	0.01	1.24	45.61	30.64	51.9	119.56	129.91	120.28	130.39	64.71
0.5	0.05	1.3	46.8	31.05	53.85	120.6	132.4	121.4	132.95	66.22

Tabela A.8: Tempos de ativação para o código de Sander Land.

Δx	Δt	P1	P2	P3	P4	P5	P6	P7	P8	P9
0.1	0.005	1.23	32.29	9.12	34.65	30.51	47.19	33.07	48.73	22.15
0.1	0.01	1.23	32.43	9.14	34.76	30.57	47.24	33.11	48.79	22.18
0.1	0.05	1.23	32.29	9.12	34.65	30.51	47.19	33.07	48.73	22.15
0.2	0.005	1.23	35.97	12.15	40.72	41.44	59.54	45.35	62.61	29.36
0.2	0.01	1.23	36.0	12.16	40.76	41.47	59.6	45.39	62.66	29.39
0.2	0.05	1.23	35.95	12.2	40.74	41.59	59.67	45.51	62.74	29.45
0.5	0.005	1.23	59.96	29.18	69.92	123.85	139.32	127.32	141.24	69.0
0.5	0.01	1.23	59.96	29.18	69.92	123.85	139.32	127.32	141.24	69.0
0.5	0.05	1.23	60.17	29.14	70.13	123.69	139.32	127.2	141.27	69.07

Tabela A.9: Tempos de ativação para o código de Alan Benson.

Δx	Δt	P1	P2	P3	P4	P5	P6	P7	P8	P9
0.1	0.005	0.95	32.05	8.94	33.25	30.16	43.26	31.6	44.17	20.02
0.1	0.01	0.94	32.01	8.93	33.21	30.15	43.24	31.6	44.15	20.0
0.2	0.005	-1.0	34.4	12.17	35.94	40.6	53.28	42.46	54.28	24.53
0.2	0.01	-1.0	34.38	12.17	35.92	40.59	53.34	42.46	54.35	24.54
0.2	0.05	-1.0	34.4	12.15	36.0	40.65	54.2	42.65	55.25	24.86
0.5	0.005	0.95	45.15	33.2	53.55	121.08	131.51	122.44	132.42	61.65
0.5	0.01	0.94	45.12	33.2	53.56	121.06	131.53	122.42	132.44	61.67
0.5	0.05	0.9	45.1	33.15	53.8	120.95	131.85	122.35	132.8	61.83

Tabela A.10: Tempos de ativação para o código de OpenCMISS.

Δx	Δt	P1	P2	P3	P4	P5	P6	P7	P8	P9
0.1	0.005	1.22	32.41	9.13	34.73	30.54	47.19	33.08	48.74	22.15
0.1	0.01	1.23	32.43	9.14	34.76	30.57	47.24	33.11	48.78	22.18
0.1	0.05	1.25	32.35	9.15	34.7	30.55	47.25	33.1	48.8	22.2
0.2	0.005	1.23	35.97	12.16	40.73	41.44	59.55	45.36	62.61	29.36
0.2	0.01	1.23	36.01	12.17	40.76	41.47	59.6	45.39	62.67	29.39
0.2	0.05	1.25	36.0	12.2	40.75	41.6	59.7	45.5	62.75	29.45
0.5	0.005	1.23	59.97	29.18	69.93	123.85	139.32	127.32	141.24	69.01
0.5	0.01	1.23	59.98	29.18	69.96	123.82	139.33	127.29	141.25	69.02
0.5	0.05	1.25	60.2	29.15	70.15	123.7	139.35	127.2	141.3	69.1

Tabela A.11: Tempos de ativação para o código de Elizabeth M. Cherry e Flavio H. Fenton.

Δx	Δt	P1	P2	P3	P4	P5	P6	P7	P8	P9
0.1	0.005	1.22	32.5	9.63	33.79	30.87	43.94	32.4	44.91	20.68
0.1	0.01	1.23	32.47	9.64	33.76	30.87	43.93	32.4	44.9	20.67
0.1	0.04	1.26	32.37	9.68	33.64	30.93	44.06	32.51	45.0	20.65
0.2	0.005	1.23	34.99	13.5	36.72	41.97	54.55	43.95	55.62	25.71
0.2	0.01	1.23	34.99	13.5	36.72	41.97	54.63	43.96	55.7	25.74
0.2	0.05	1.27	35.08	13.57	36.85	42.13	55.56	44.22	56.68	26.14
0.5	0.005	1.23	46.51	44.81	61.71	133.61	142.95	135.01	143.96	72.66
0.5	0.01	1.23	46.49	44.82	61.75	133.59	142.97	134.99	143.98	72.69
0.5	0.05	1.27	46.51	44.95	62.2	133.66	143.42	135.08	144.46	73.04

Tabela A.12: Tempos de ativação para o código FEniCS/PyCC.

Δx	Δt	P1	P2	P3	P4	P5	P6	P7	P8	P9
0.1	0.005	1.3	30.98	7.87	31.47	25.37	37.64	25.99	40.4	19
0.1	0.01	0.95	31.2	7.9	31.7	25.5	37.93	26.13	38.08	17.8
0.1	0.05	1.0	33.1	8.25	33.7	26.65	40.45	27.4	40.7	19.05
0.2	0.005	1.3	30.54	8.43	30.55	28.19	37.66	28.43	36.5	17
0.2	0.01	0.95	30.72	8.46	30.74	28.27	37.84	28.51	37.59	17.11
0.2	0.05	1.0	32.3	8.7	32.4	29.0	39.45	29.35	39.25	18.0
0.5	0.005	1.3	33.77	16.37	33.05	53.32	60.22	51.92	27.8	14
0.5	0.01	0.95	33.88	16.39	33.17	53.37	60.3	51.98	58.28	27.96
0.5	0.05	1.0	34.95	16.65	34.25	53.95	61.1	52.6	59.15	28.45

Tabela A.13: Tempos de ativação para o código de Oliveira (2013).

Δx	Δt	P1	P2	P3	P4	P5	P6	P7	P8	P9
0.1	0.005	1.75	35.05	5.7	35.4	25.025	42.2	25.625	42.475	20.2
0.1	0.01	1.75	35.4	5.75	35.8	25.2	42.65	25.8	42.95	20.45
0.1	0.05	2.0	38.25	6.25	38.5	26.75	46	27.25	46.25	22.25
0.2	0.005	2.2	37.925	6.875	38.325	31.4	47.3	32.075	47.55	22.75
0.2	0.01	2.25	38.2	6.95	38.6	31.5	47.6	32.2	47.85	22.9
0.2	0.05	2.5	40.5	7.25	41	32.25	50	33.0	50.25	24.25
0.5	0.005	6.45	52.0	12.65	52.5	69.725	83.975	70.05	84.125	37.55
0.5	0.01	6.5	52.15	12.7	52.65	69.8	84.15	70.15	84.3	37.65
0.5	0.05	6.75	53.0	13.0	53.5	70.25	85.5	70.5	85.75	38.5

Referências Bibliográficas

Amorim, R. M., dos Santos, R. W., 2013. Solving the cardiac bidomain equations using graphics processing units. Journal of Computational Science 4, 370–376.

Bartocci, E., Cherry, E. M., Glimm, J., Grosu, R., Smolka, S. A., Fenton, F. H., 2011. Toward real-time simulation of cardiac dynamics. In: CMSB11: Proceedings of the 9th International Conference on Computational Methods in Systems Biology. pp. 103–112.

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

Bernaschi, M., Fatica, M., Melchionna, S., Succi, S., Kaxiras, E., 2010. A flexible high-performance Lattice Boltzmann GPU code for the simulations of fluid flows in complex geometries. Concurrency and Computation: Practice and Experience 22 (1), 1–14.

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

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

Bouzidi, M., Firdaouss, M., Lallemand, P., 2001. Momentum transfer of a boltzmann-lattice fluid with boundaries. Physics of Fluids 13(11).

Campos, F. O., 2008. Modelagem computacional da eletrofisiologia cardíaca: o desenvolvimento de um novo modelo para células de camundongos e a avaliação de novos esquemas numéricos. Master's thesis, Universidade Federal de Juiz de Fora.

Campos, J., Oliveira, R., dos Santos, R., Rocha, B., 2015. Lattice Boltzmann method for parallel simulations of cardiac electrophysiology using GPUs. Journal of Computational and Applied Mathematics.

Campos, J. O., 2013. Simulações computacionais de escoamento de fluidos e problemas de reação-difusão através do método de lattice Boltzmann.

Constanzo, L., 2007. Fisiologia, 3rd Edition. Elsevier.

Dawson, S. P., Chen, S., D., D. G., 1992. Lattice Boltzmann computations for reaction-diffusion equations. Journal of Chemical Physics 98, 1514–1523.

Eymard, R., Gallouët, T., Herbin, R., 2000. Finite volume methods. Handbook of numerical analysis 7, 713–1018.

Ginzburg, I., 2005. Equilibrium-type and link-type lattice Boltzmann models for generic advection and anisotropic-dispersion equation. Advances in Water Resources.

Giraldi, G. A., Xavier, A. V., Jr., A. L. A., Rodrigues, P. S., 2005. Lattice gas cellular automata for computational fluid animation. CoRR abs/cs/0507012.

Golbert, D. R., 2009. Modelos de lattice-Boltzmann aplicados a simulação computacional do escoamento de fluidos incompressíveis. Master's thesis, Laboratório Nacional de Computação Científica.

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

Habich, J., Feichtinger, C., Köstler, H., Hager, G., Wellein, G., 2011. Performance engineering for the lattice Boltzmann method on gpgpus: Architectural requirements and performance results. CoRR abs/1112.0850.

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

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

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

Keener, J., Sneyd, J., 1998. Mathematical Physiology. Springer.

Latt, J., 2007. How to implement your DdQq dynamics with only q variables per node (instead of 2q). Tech. rep., Tufts University.

LeVeque, R., 2007. Finite difference methods for ordinary and partial differential equations. Siam.

Lloyd, C. M., Lawson, J. R., Hunter, P. J., Nielsen, P. F., Sep. 2008. The CellML Model Repository. Bioinformatics 24 (18), 2122–2123.

Luo, C., Rudy, Y., 1991. A model of the ventricular cardiac action potential. depolarization, repolarization, and their interaction. Circulation Research 68 (6), 1501–26.

Mahajan, A., Shiferaw, Y., Sato, D., Baher, A., Olcese, R., Xie, L.-H., Yang, M.-J., Chen, P.-S., Restrepo, J. G., Karma, A., Garfinkel, A., Qu, Z., Weiss, J. N., 2008. A rabbit ventricular action potential model replicating cardiac dynamics at rapid heart rates. Biophysical Journal 94, 392–410.

Mattila, K., Hyvaluoma, J., Rossi, T., Aspnas, M., Westerholm, J., 2007. An efficient swap algorithm for the lattice Boltzmann method. Computer Physics Communications 176, 200–210.

Mirams, G. R., Arthurs, C. J., Bernabeu, M. O., Bordas, R., Cooper, J., Corrias, A., Davit, Y., Dunn, S.-J., Fletcher, A. G., Harvey, D. G., Marsh, M. E., Osborne, J. M., Pathmanathan, P., Pitt-Francis, J., Southern, J., Zemzemi, N., Gavaghan, D. J., 2013. Chaste: An open source C++ library for computational physiology and biology. PLoS Computational Biology.

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

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

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

Neumann, P., Neckel, T., 2013. A dynamic mesh refinement technique for Lattice Boltzmann simulations on octree-like grids. Comput. Mech. 51, 237–253.

Niederer, S. A., Kerfoot, E., Benson, A. P., Bernabeu, M. O., Bernus, O., Bradley, C., Cherry, E. M., Clayton, R., Fenton, F. H., Garny, A., Heidenreich, E., Land, S., Maleckar, M., Pathmanathan, P., Plank, G., Rodríguez, J. F., Roy, I., Sachse, F. B., Seemann, G., Skavhaug, O., Smith, N. P., 2011. Verification of cardiac tissue electrophysiology simulators using an N-version benchmark. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences 13, 4331–4351.

NVIDIA, Julho 2013. GPU accelerated computing with C and C++. Publicação Eletrônica: https://developer.nvidia.com/get-started-cuda-cc.

Oliveira, R. S., 2013. Algoritmos paralelos e adaptativos no tempo e no espaço para simulação numérica da eletrofisiologia do coração. Ph.D. thesis, Universidade Federal de Minas Gerais.

Oliveira, R. S., Rocha, B. M., Burgarelli, D., Meira, W., Weber do Santos, R., 2012. A parallel accelerated adaptive mesh algorithm for the solution of electrical models of the heart. International Journal of High Performance Systems Architecture 4 (2).

Pacheco, P. S., 2011. An Introduction to Parallel Programming, 1st Edition. Elsevier, San Francisco.

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

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

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

Rocha, B. M., Campos, F. O., Amorim, R. M., Plank, G., dos Santos, R. W., Liebmann, M., Haase, G., 2011. Accelerating cardiac excitation spread simulations using graphics processing units. Concurrency and Computation: Practice and Experience 23, 708–720.

Rush, S., Larsen, H., 1978. A practical algorithm for solving dynamic membrane equations. IEEE Transactions on Biomedical Engineering 25, 389–392.

Sachse, F. B., 2004. Computational Cardiology. Springer Verlag.

SoBiologia, Julho 2015. Tecido muscular estriado cardíaco. Publicação Eletrônica: https://http://www.sobiologia.com.br/conteudos/Histologia/epitelio24.php.

Succi, S., 2013. The Lattice Boltzmann Equation: For Fluid Dynamics and Beyond. Oxford Science Publications.

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

ten Tusscher, K. H. W. J., Panfilov, A. V., 2006. Alternans and spiral breakup in a human ventricular tissue model. Am J Physiol Heart Circ Physiol 291, H1088–H1100.

Trefethen, L. N., Bau, D., 1997. Numerical Linear Algebra. SIAM.

Vetter, F. J., McCulloch, A. D., 1998. Three-dimensional analysis of regional cardiac function: a model of rabbit ventricular anatomy. Progress in Biophysics and Molecular Biology 69, 157–183.

Vigmond, E., Plank, M. H. G., Leon, L., 2003. Computational tools for modeling electrical activity in cardiac tissue. J. Electrocardiol. 36, 69–74.

Wang, W., Xu, L., Cavazos, J., Huang, H. H., Kay, M., 2014. Fast acceleration of 2D wave propagation simulations using modern computational accelerators. PLoS One 9 (1), e86484.

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

Yoshida, H., Nagaoka, M., 2010. Multiple-relaxation-time lattice Boltzmann model for the convection and anisotropic diffusion equation. Journal of Computational Physics 229, 7774–7795.

Zhang, X., Bengough, A. G., Crawford, J. W., Young, I. M., 2002. A lattice BGK model for advection and anisotropic dispersion equation. Advances in Water Resources 25, 1–8.

Zou, Q., He, X., 1997. On pressure and velocity boundary conditions for the lattice Boltzmann BGK model. American Institute of Physics 9, 1591–1598.