

**ALGORITMOS PARALELOS E ADAPTATIVOS
NO TEMPO E NO ESPAÇO PARA SIMULAÇÃO
NUMÉRICA DA ELETROFISIOLOGIA DO
CORAÇÃO**

RAFAEL SACHETTO OLIVEIRA

ALGORITMOS PARALELOS E ADAPTATIVOS
NO TEMPO E NO ESPAÇO PARA SIMULAÇÃO
NUMÉRICA DA ELETROFISIOLOGIA DO
CORAÇÃO

Tese apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Doutor em Ciência da Computação.

ORIENTADOR: WAGNER MEIRA JÚNIOR
COORIENTADOR: RODRIGO WEBER DOS SANTOS

Belo Horizonte

22 de fevereiro de 2013

© 2013, Rafael Sachetto Oliveira.
Todos os direitos reservados.

Oliveira, Rafael Sachetto

B557e Algoritmos paralelos e adaptativos no tempo e no
espaço para simulação numérica da eletrofisiologia do
coração / Rafael Sachetto Oliveira. — Belo Horizonte,
2013

xxiii, 103 f. : il. ; 29cm

Tese (doutorado) — Universidade Federal de Minas
Gerais. Departamento de Ciência da Computação.

Orientador: Wagner Meira Júnior

Coorientador: Rodrigo Weber dos Santos

1. Computação-Teses. 2. Computação Paralela-Teses.
3. Sistemas Distribuídos-Teses. 4. Eletrofisiologia
Cardíaca. I. Orientador. II. Corientador. III. Título.

CDU 519.6*73 (043)



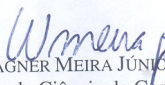
UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

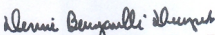
FOLHA DE APROVAÇÃO

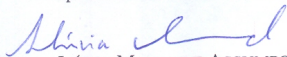
Algoritmos paralelos e adaptativos no tempo e no espaço
para simulação numérica da eletrofisiologia do coração

RAFAEL SACHETTO OLIVEIRA

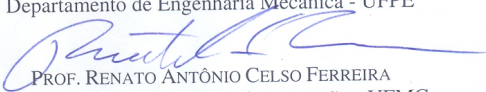
Tese defendida e aprovada pela banca examinadora constituída pelos Senhores:

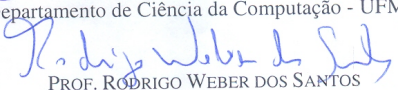

PROF. WAGNER MEIRA JÚNIOR - Orientador
Departamento de Ciência da Computação - UFMG


PROFA. DENISE BURGARELLI DUCZMAL
Departamento de Matemática - UFMG


PROFA. LÚCIA MARIA DE ASSUMPÇÃO DRUMMOND
Departamento de Ciência da Computação - UFF


PROF. PAULO ROBERTO MACIEL LYRA
Departamento de Engenharia Mecânica - UFPE


PROF. RENATO ANTÔNIO CELSO FERREIRA
Departamento de Ciência da Computação - UFMG


PROF. RODRIGO WEBER DOS SANTOS
Departamento de Ciência da Computação - UFJF

Belo Horizonte, 22 de fevereiro de 2013.

Dedico essa tese a todos que, de alguma forma, me ajudaram nessa grande conquista.

*“I used to want to change the world.
Now I just want to leave the room with a little dignity.”*
(Lotus Weinstock)

Resumo

Modelos computacionais se tornaram ferramentas importantes para o estudo do comportamento elétrico do coração. Porém, a alta complexidade dos processos biofísicos envolvidos se traduz em modelos computacionalmente caros. Neste trabalho propusemos, implementamos e combinamos estratégias de computação paralela utilizando unidade de processamento gráfico (GPU), malhas adaptativas no espaço e métodos adaptativos no tempo com o objetivo de diminuir o tempo de execução das simulações da eletrofisiologia cardíaca.

Na primeira abordagem, três diferentes implementações GPU foram comparadas, OpenGL, NVIDIA CUDA e OpenCL, a uma implementação paralela em CPU utilizando OpenMP. A abordagem utilizando OpenGL mostrou ser a mais rápida com um *speedup* de 446 (em comparação com a implementação utilizando uma cpu com 4 núcleos) para a solução do sistema não-linear de EDOs associada à solução do modelo cardíaco, enquanto a implementação em CUDA foi a mais rápida para o solução numérica da EDP parabólica atingindo um *speedup* de 8. Apesar de apresentarmos uma melhoria significativa no tempo de execução das simulações da eletrofisiologia cardíaca, ainda nos encontramos bem distantes de simulações em tempo real.

O uso da malha adaptativa *Autonomous Leaves Graph* (ALG) também se mostrou bastante atrativo, pois a frente de onda elétrica que se propaga corresponde a uma pequena fração do tecido cardíaco. Usualmente, a resolução numérica das equações diferenciais parciais que modelam o fenômeno requer discretizações espaciais suficientemente finas para acompanhar a frente de onda, que é da ordem de $0,25\text{ mm}$. O uso de malhas uniformes leva a um custo computacional alto, pois exige um número grande de pontos de malha. Neste sentido, os testes relatados nesta tese mostram que simulações de modelos bidimensionais do tecido cardíaco foram aceleradas em até 80 vezes pelo uso do algoritmo de malhas adaptativas, sem perda na precisão da solução numérica obtida.

Além disso, desenvolvemos uma versão paralela do método de Euler com passo de tempo adaptativo, tanto utilizando unidades de processamento com múltiplos nú-

cleos quanto GPUs. Por fim, combinamos os métodos paralelos desenvolvidos com as técnicas de malhas adaptativas. Com a combinação dessas técnicas alcançamos taxas de aceleração muito superiores às encontradas em trabalhos recentes, atingindo mais de 1.700 vezes de aceleração utilizando apenas uma máquina equipada com 8 processadores e uma GPU.

Palavras-chave: Computação Paralela, Eletrofisiologia Cardíaca, Malhas adaptativas.

Abstract

Computational models have become valuable tools for the study of the electrical behavior of the heart. However, the high complexity of the biophysical processes translates into expensive computational models. In this work we propose, implement and combine strategies using parallel computing and graphical processing unit (GPU), adaptive meshes in space and time adaptive methods in order to reduce the execution time of cardiac electrophysiology simulations.

In our first approach different GPU implementations are compared, OpenGL, NVIDIA CUDA and OpenCL, to a CPU multicore implementation that uses OpenMP. The OpenGL approach showed to be the fastest with a speedup of 446 (compared to the multicore implementation) for the solution of the nonlinear system of ordinary differential equations (ODEs) associated to the solution of the cardiac model, whereas CUDA was the fastest for the numerical solution of the parabolic partial differential equation with a speedup of 8. Although we achieved a significant improvement in the execution time of the cardiac electrophysiology simulations, we are far from real-time simulations.

The use of the Autonomous Leaves Graph (ALG) adaptive mesh also proved very attractive because the electrical wavefront that propagates corresponds only to a small fraction of the cardiac tissue. Usually, the numerical solution of the partial differential equations that model the phenomenon requires fine enough space discretizations to follow the wave front, which is about 0.25 mm . The use of uniform meshes leads to a high computational cost, as it requires a large number of mesh points. In this sense, the tests reported in this work show that the bi-dimensional heart tissue model simulations were accelerated up to 80 times by using the adaptive mesh algorithm, with no loss in accuracy of the numerical solution obtained.

Furthermore, we developed a parallel version of the Euler method with adaptive time step, using both multi-core or GPUs. Finally, we combine the developed parallel methods with the adaptive meshes techniques. The combination of these techniques allowed to achieve acceleration rates much higher than those found in recent work,

reaching over $1.700\times$ acceleration using only a machine equipped with 8 processors and a single GPU.

Lista de Figuras

2.1	Anatomia do coração (adaptado de Netter [2008]).	8
2.2	Visão da membrana celular (bicamada fosfolipídica e proteínas que a permeiam).	9
2.3	Ilustração esquemática de um potencial de ação medido na membrana de um miócito cardíaco (adaptado de [Sachse, 2004]).	10
2.4	A Membrana celular e sua aproximação por um circuito resistor capacitor.	12
2.5	Onda espiral gerada pelo modelo monodomínio utilizando volumes finitos e o modelo celular de C. Luo & Yoram Rudy [1991]	24
4.1	Algoritmo para a solução do modelo monodomínio.	32
4.2	<i>Pipeline</i> Gráfica moderna. Tanto os processadores de vértices quanto o de fragmentos são programáveis.	34
4.3	Esquema da interação entre CPU e GPU para o <i>loop</i> do GCP.	35
4.4	Organização das <i>threads</i> em um programa CUDA.	36
4.5	Quadrado unitário e ligações do grafo (adaptado de Burgarelli et al. [2006]).	42
4.6	Refinamento da célula noroeste. (adaptado de Burgarelli et al. [2006]). . .	44
4.7	Seleção do cacho (A), desrefino do cacho selecionado (B) e conexão dos nós (C) (adaptado de Burgarelli et al. [2006]).	44
4.8	Simplificações após o refinamento (adaptado de Burgarelli et al. [2006]). . .	45
4.9	Malha após o desrefinamento de uma célula (adaptado de Burgarelli et al. [2006]).	46
4.10	Exemplo de malha refinada usando ALG	47
4.11	Simulação de propagação da onda elétrica (distribuição do potencial transmembrânico V_m) em tecido computacional do ventrículo em 30 ms (cima), 100 ms (meio) e 260 ms (baixo) utilizando o esquema de adaptatividade do ALG.	53

4.12	Comparação do potencial transmembrânico V ao longo do tempo em um nó da malha fixa e da malha adaptativa usando o esquema de adaptatividade do ALG.	54
4.13	Atualização do vetor <code>cellsToSolve</code> e do mapa de livres após o desrefinamento do cacho nordeste de uma malha ALG.	68
5.1	Simulação de propagação da onda elétrica (distribuição do potencial transmembrânico V_m) em tecido computacional tridimensional do coração de um coelho utilizando o esquema de adaptatividade do ALG.	75
A.1	Fluxo, potenciais e concentrações iônicas da equação de Nernst (adaptada de [Sachse, 2004]).	88
A.2	Fluxo, potenciais e concentrações iônicas da equação de Goldman-Hodgkin-Katz (adaptada de [Sachse, 2004]).	90
A.3	Variação da condutividade de sódio (G_{Na}) em função do tempo após uma mudança no potencial transmembrânico (Adaptado de [Keener & Sneyd, 1998]).	93
A.4	Potencial de ação do modelo de Hodgkin & Huxley [1952] (adaptado de Keener & Sneyd [1998]).	95
A.5	Variação das condutividades g_{na} e g_k do modelo Hodgkin-Huxley durante um potencial de ação (adaptado de Keener & Sneyd [1998]).	95
A.6	Representação esquemática das correntes e bombas incluídas no modelo C. Luo & Yoram Rudy [1991].	98
A.7	Representação esquemática das correntes e bombas incluídas no modelo Ten Tusscher et al. [2004].	99

Lista de Tabelas

4.1	Diretivas de <i>Kernel</i> em CUDA C e OpenCL	36
4.2	Detalhes da preparação <i>in-silico</i> dos tecidos e propriedades das malhas . .	37
4.3	Comparação entre as implementações CPU e GPU para o problema de EDOs. Os tempos de execução são dados em segundos.	40
4.4	Comparação entre as implementações CPU e GPU para o problema parabólico usando tanto o formato CSR quanto o DIA. Os tempos de execução são dados em segundos.	41
4.5	Erros e taxas de aceleração obtidas com a utilização de malhas grosseiras .	51
4.6	Erros e taxas de aceleração obtidas com a utilização de malhas adaptativas	52
4.7	Tempos totais de montagem de matriz e execução do GC para 3 simulações distintas	52
4.8	Erros e taxas de aceleração obtidas com a utilização de malhas adaptativas e método adaptativo no tempo para a solução das EDOs.	58
4.9	Taxas de aceleração obtidas com a utilização de malhas adaptativas e método adaptativo no tempo para a solução das EDO em relação somente a utilização de malhas adaptativas.	59
4.10	Resultados da execução paralela (8 <i>threads</i>) de uma simulação utilizando uma malha adaptativa com h mínimo de $50 \mu m$ e de máximo $200 \mu m$ e método de Euler adaptativo no tempo.	63
4.11	Resultados da execução paralela (8 <i>threads</i>) de uma simulação utilizando uma malha adaptativa com h mínimo de $50 \mu m$ e de máximo $200 \mu m$, método de Euler adaptativo no tempo e malha com $102.400 \mu m$	64
4.12	Resultados da execução paralela (8 <i>threads</i> + GPU) de uma simulação utilizando uma malha adaptativa com h mínimo de $50 \mu m$ e de máximo $200 \mu m$ e método de Euler adaptativo no tempo.	69

4.13	Resultados da execução paralela (8 <i>threads</i> + GPU) de uma simulação utilizando uma malha adaptativa com h mínimo de $50\ \mu m$ e de máximo $200\ \mu m$, método de Euler adaptativo no tempo e e malha num domínio de $102.400\mu m$ x $102.400\mu m$	69
4.14	Taxas de aceleração das técnicas desenvolvidas	70

Lista de Algoritmos

1	Passos para a solução numérica do modelo monodomínio	48
2	Trecho de código com a implementação paralela da resolução dos sistemas de EDOs com passo de tempo fixo	61
3	Trecho de código com a implementação paralela da resolução dos sistemas de EDOs com passo de tempo adaptativo	61
4	Implementação paralela do método GC utilizando ALG	63
5	Passos para a solução numérica do modelo monodomínio	67

Sumário

Resumo	xi
Abstract	xiii
Lista de Figuras	xv
Lista de Tabelas	xvii
1 Introdução	1
1.1 Motivação	1
1.2 Argumento da tese	2
1.3 Contribuições da tese	3
1.4 Outras contribuições	4
1.5 Organização do texto	5
2 Modelagem da Eletrofisiologia Cardíaca	7
2.1 Introdução	7
2.2 Fisiologia da Membrana Celular	9
2.3 Potencial de Ação e Excitabilidade	10
2.4 Modelo Matemático para a Membrana Celular	11
2.5 Modelos para o tecido	12
2.5.1 O modelo bidomínio	13
2.5.2 O modelo monodomínio	15
2.6 Discretização temporal e operador <i>splitting</i>	16
2.7 O método dos volumes finitos	17
2.7.1 O MVF aplicado ao modelo monodomínio	18
2.8 Simulando a eletrofisiologia cardíaca	23
3 Estratégias para aceleração das simulações da eletrofisiologia cardíaca	25

3.1	Computação paralela	25
3.2	Utilização de GPUs	26
3.3	Utilização de malhas adaptativas	26
3.3.1	Trabalhos relacionados	27
3.4	Utilização de passo de tempo adaptativo	29
3.5	Combinando as estratégias de aceleração	29
4	Implementações e resultados obtidos	31
4.1	Utilização de GPUs	31
4.1.1	Implementações	31
4.1.2	Métodos	37
4.1.3	EDOs e o problema parabólico	37
4.1.4	Erro numérico	38
4.1.5	Resultados	38
4.1.6	Discussão	40
4.2	Utilização da malha adaptativa ALG	42
4.2.1	A estrutura de dados ALG	42
4.2.2	Refinamento da malha	43
4.2.3	Desrefinamento da malha	44
4.2.4	ALG e o problema do monodomínio	46
4.2.5	Configuração dos experimentos	49
4.2.6	Resultados	51
4.2.7	Discussão	54
4.3	Método adaptativo no tempo	54
4.3.1	Descrição e implementação	55
4.3.2	Configuração dos experimentos	57
4.3.3	Resultados	58
4.3.4	Discussão	59
4.4	Implementações paralelas em CPU	59
4.4.1	Descrição e implementação	59
4.4.2	Configuração dos experimentos	62
4.4.3	Resultados	62
4.4.4	Discussão	64
4.5	Implementação paralela utilizando GPU	65
4.5.1	Descrição e implementação	65
4.5.2	Resultados	67
4.5.3	Discussão	69

5	Conclusões e trabalhos futuros	73
5.1	Conclusões	73
5.2	Trabalhos em andamento e futuros	74
5.2.1	Paralelização das operações da malha ALG	75
5.2.2	Implementação da malha adaptativa ALG utilizando <i>clusters</i> de GPUs	75
5.2.3	Utilização de preconditionadores	76
5.2.4	Balanceamento de carga na GPU	76
5.2.5	Desenvolvimento de diferentes estratégias para a decisão de refinamento/desrefinamento no ALG	76
	Referências Bibliográficas	79
	Apêndice A Modelagem da eletrofisiologia cardíaca	87
A.1	Potenciais de Equilíbrio da Membrana Celular	87
A.1.1	O Potencial de Nernst	87
A.1.2	Equação de Goldman-Hodgkin-Katz	89
A.2	Modelos para a Corrente Iônica	91
A.3	Canais Iônicos	92
A.3.1	O Modelo de Dois Estados	92
A.3.2	O Modelo de Subunidades	93
A.4	O Modelo de Hodgkin e Huxley	94
A.5	O Modelo Luo Rudy	97
A.6	O Modelo de Ten Tusscher	98
A.7	Aplicação do MVF para a resolução do monodomínio	100
A.7.1	MVF com ALG	101

Capítulo 1

Introdução

1.1 Motivação

As doenças cardíacas são responsáveis por um terço do total de mortes no mundo [OMS, 2010]. Acredita-se que mais de 300 mil pessoas morrem por ano no Brasil vítimas de anomalias relacionadas principalmente à atividade elétrica do coração. O conhecimento sobre a eletrofisiologia cardíaca é fundamental para a compreensão de muitos aspectos do comportamento fisiológico e fisiopatológico do coração. A eletrofisiologia está fortemente acoplada à deformação mecânica que faz com que o coração exerça a função de bomba de sangue.

Modelos computacionais [Hodgkin & Huxley, 1952; Plonsey, 1988] se tornaram ferramentas importantes para o estudo e a compreensão de fenômenos com esse nível de complexidade, por permitirem que informações coletadas a partir de experimentos em diferentes escalas físicas sejam combinadas para se obter um entendimento melhor da funcionalidade do sistema como um todo. Como já era de se esperar, a alta complexidade dos processos biofísicos se traduz em complexos modelos matemáticos e computacionais. Os modelos cardíacos modernos são descritos por sistemas não-lineares de equações diferenciais parciais com milhões de variáveis e centenas de parâmetros.

O modelo bidomínio [Plonsey, 1988] é considerado uma das mais completas descrições da atividade elétrica do tecido cardíaco. Utilizando hipóteses apropriadas as equações do bidomínio podem ser reduzidas para um modelo mais simples, denominado monodomínio, o qual é menos intensivo computacionalmente [Sundnes, 2006]. Infelizmente, as simulações em larga escala, tais como as resultantes da discretização de um coração inteiro, permanecem como um desafio computacional. Apesar das dificuldades e da complexidade associadas à implementação e uso desses modelos, os benefícios e aplicações justificam seu uso. Os modelos computacionais têm sido utilizados durante

testes de novos fármacos, desenvolvimento de novos dispositivos médicos e novas técnicas de diagnóstico não-invasivo para diversas doenças cardíacas [Gima & Rudy, 2002; Weber dos Santos et al., 2004b].

Nas simulações da eletrofisiologia cardíaca, a frente de onda que viaja pelo coração é muito fina, refletindo em uma variação rápida no potencial transmembrânico. Esta flutuação faz com que a resolução numérica das equações diferenciais parciais que modelam o fenômeno requeiram discretizações espaciais suficientemente finas para acompanhar a frente de onda, que é da ordem de $0,2\text{ mm}$ [Hunter & Borg, 2003], a fim de assegurar resultados suficientemente precisos. A execução de simulações cardíacas em malhas com um grande número de nós tem um custo computacional alto, pois exige a solução repetida de sistemas lineares com milhões de graus de liberdade. Os requisitos de memória das simulações também representam um problema, pois tornam-se cada vez maiores. A utilização de métodos de malha adaptativa fornece uma solução para esses dois problemas. Manter a resolução extremamente fina apenas onde é necessário (isto é, nas proximidades da frente de onda) reduz significativamente o número de graus de liberdade das variáveis envolvidas, resultando em cálculos mais rápidos, um uso de memória menor e na diminuição da necessidade de uso de disco para a gravação de arquivos de saída [Southern et al., 2010].

Outra característica a ser considerada é a dificuldade de resolver os sistemas de equações relacionados aos modelos celulares, pois estes são complexos, de natureza altamente não-linear e envolvem múltiplas escalas. Isso faz com que a simulação destes modelos demande alto poder computacional além de muito tempo, variando de acordo com a complexidade do fenômeno. Para resolver esse problema podemos utilizar métodos numéricos que são capazes de adaptar o passo de tempo, como em Campos et al. [2011]. Esta adaptação de tempo tem como objetivo reduzir o tempo de execução das soluções dos sistemas de equações diferenciais ordinárias. Isso é feito aumentando o passo de tempo em regiões onde o limite de estabilidade permite. Entretanto, em regiões onde a física e/ou o refinamento da malha exigem é necessário manter o passo de tempo com valores menores, a fim de garantir que os percentuais de erro permaneçam em patamares aceitáveis.

1.2 Argumento da tese

Dada a importância do estudo e simulação de fenômenos complexos, como os citados na seção anterior, e as limitações das atuais soluções (desempenho, escalabilidade, etc.) podemos ver que existem problemas a serem solucionados. A hipótese deste

trabalho é que é possível realizar simulações cardíacas mais eficientes tanto em tempo de execução quanto em utilização de memória principal e secundária pela adoção de novas estratégias de solução utilizando métodos numéricos adaptativos no tempo e no espaço, além de técnicas de computação paralela.

Nossa estratégia é então utilizar computação paralela, tecnologias emergentes como computação de propósito geral em unidades de processamento gráfico (GPGPU) [GPGPU, 2010], técnicas como malhas adaptativas e métodos numéricos adaptativos no tempo, a fim de aumentar a escalabilidade e o desempenho das simulações da eletrofisiologia cardíaca, mantendo a precisão numérica, visando a criação de ferramentas que aceleram as simulações dos fenômenos relacionados à eletrofisiologia cardíaca.

1.3 Contribuições da tese

Como principais contribuições desta tese podemos citar o desenvolvimento de um algoritmo adaptativo no tempo paralelizado em GPU, utilização de uma nova técnica de malha adaptativa para a simulação da eletrofisiologia cardíaca e principalmente a combinação de técnicas de computação paralela (memória compartilhada e GPU), métodos adaptativos no tempo para a resolução de EDOs e métodos de malha adaptativa que conjuntamente aceleraram as simulações em mais de 1.700 vezes, para a classe de problemas analisada. Vale ressaltar que as taxas de aceleração obtidas foram consideravelmente maiores que as encontradas em trabalhos que utilizam técnicas semelhantes aplicadas a simulação da eletrofisiologia cardíaca.

Alguns artigos foram publicados durante a evolução deste trabalho. Esses trabalhos introduzem alguns conceitos ou modelos descritos neste texto. O primeiro trabalho relacionado com esta tese é a dissertação de mestrado do aluno [Oliveira, 2008] onde o mesmo introduz alguns conceitos de modelagem da eletrofisiologia cardíaca. O segundo trabalho é um artigo publicado na conferência *International Conference on Parallel Processing and Applied Mathematics* (PPAM '2011) [Oliveira et al., 2011]. Nesse artigo, o aluno investiga técnicas de programação em GPU para a aceleração das simulações da eletrofisiologia cardíaca. O objetivo deste trabalho é estudar o desempenho de diferentes interfaces de programação para GPU para a simulação da eletrofisiologia cardíaca. Após essa análise, a implementação que apresentou o melhor custo-benefício (melhor desempenho/facilidade de implementação) foi utilizada nesta tese.

No segundo artigo, publicado na conferência *The 12th International Conference on Computational Science and Its Applications* (ICCSA '2012) [Oliveira et al., 2012a],

o aluno descreve a adaptação e formulação de um método de malha adaptativa, descrito em Burgarelli et al. [2006], para as simulações da eletrofisiologia cardíaca utilizando o modelo monodomínio. Vale ressaltar que este foi o primeiro artigo, de acordo com nossas pesquisas bibliográficas, a utilizar essa malha adaptativa para a simulação da eletrofisiologia cardíaca utilizando o modelo monodomínio. O aluno então estendeu este artigo, paralelizando a implementação computacional para ambientes de memória compartilhada, com o objetivo de reduzir o tempo global para a solução do problema. Este trabalho foi publicado no periódico *International Journal of High Performance Systems Architecture* [Oliveira et al., 2012b]. Além disso, o aluno também publicou no periódico *Computational and Mathematical Methods in Medicine* [Barros et al., 2012] o trabalho que mostra o desenvolvimento de um modelo de eletrofisiologia cardíaca que captura a microestrutura do tecido cardíaco, utilizando uma discretização espacial muito fina ($8\mu m$). Para lidar com os desafios computacionais, o modelo foi paralelizado usando uma abordagem híbrida: a computação em *cluster* e GPGPU. Vale ressaltar que o aluno também é o primeiro autor deste artigo, juntamente com Barros.

1.4 Outras contribuições

Paralelamente o aluno também participou como coautor em outros trabalhos relacionados com seus principais tópicos de pesquisa: Computação de alto desempenho e modelagem computacional. Como resultado desta interação, alguns artigos foram publicados e estão listados abaixo:

- Xavier et al. [2009]. Publicado no periódico *International Journal of Parallel Programming*. Esse trabalho propõe um algoritmo paralelo multi-nível para a simulação da eletrofisiologia cardíaca utilizando o modelo bidomínio.
- Coutinho et al. [2009]. Publicado na conferência *21st International Symposium on Computer Architecture and High Performance Computing* (SBAC-PAD '09). Neste trabalho os autores propõem uma ferramenta para realizar *proffiling* de aplicações GPGPU.
- Teodoro et al. [2009a]. Publicado na conferência *21st International Symposium on Computer Architecture and High Performance Computing* (SBAC-PAD '09). Neste artigo os autores desenvolvem um ambiente de tempo de execução e *framework* de programação que suporta a implementação escalável e eficiente de aplicações paralelas em ambientes distribuídos e heterogêneos.

- Teodoro et al. [2009b]. Publicado na conferência *IEEE International Conference on Cluster Computing* (CLUSTER '2009). O objetivo deste artigo é investigar a utilização coordenada de CPU e GPU para melhorar a eficiência de aplicações em relação a utilização de somente um dos dispositivos de forma independente.
- Catalyurek et al. [2010]. Capítulo publicado no livro *Scientific Computing with Multicore and Accelerators* em 2010. Este capítulo de livro é uma extensão do artigo de Teodoro et al. [2009b].
- Andrade et al. [2012]. Publicado na conferência *International Conference on Computational Science* (ICCS '2012). Neste trabalho os autores desenvolvem o HASCH (*High Performance Automatic Spell CHEcker*), cujo objetivo é corrigir a ortografia em textos em português coletados da Internet.

1.5 Organização do texto

O restante deste texto está organizado da seguinte forma. No próximo capítulo apresentamos uma visão geral sobre os aspectos teóricos da modelagem de eletrofisiologia cardíaca, além dos métodos numéricos utilizados para a resolução dos modelos matemáticos empregados nas simulações da eletrofisiologia cardíaca. No capítulo 3 descrevemos as nossas propostas para a aceleração das simulações da eletrofisiologia cardíaca e no capítulo 4 mostramos os resultados obtidos com a implementação dessas propostas. Finalmente, apresentamos as conclusões deste trabalho bem como discutimos os trabalhos futuros relacionados à tese no capítulo 5.

Capítulo 2

Modelagem da Eletrofisiologia Cardíaca

2.1 Introdução

O conhecimento sobre a eletrofisiologia cardíaca é fundamental para a compreensão de muitos aspectos do comportamento fisiológico e fisiopatológico do coração. A eletrofisiologia está fortemente acoplada à deformação mecânica que faz com que o coração exerça a função de bomba de sangue.

O coração é o órgão responsável por impulsionar sangue para todo o corpo. Ele é composto por quatro câmaras, que se dividem em dois ventrículos e dois átrios. O fluxo de sangue chega ao coração pelo átrio direito, de onde é expelido para o ventrículo direito, que por sua vez impulsiona sangue para os pulmões, onde ocorrem as trocas gasosas. Posteriormente, o sangue segue para o átrio esquerdo e em seguida para o ventrículo esquerdo, que impulsiona o fluxo para os órgãos periféricos. A anatomia do coração é mostrada na figura 2.1.

Para que o coração exerça corretamente sua função, os miócitos precisam estar sincronizados. Essa sincronização é feita pela propagação rápida de uma onda elétrica por todo o órgão, o que leva à contração. A propagação desta onda é modulada por potenciais transmembrânicos resultantes de atividades elétricas nas células ou de fluxos de corrente através da membrana celular.

Vários experimentos vêm sendo realizados com o objetivo de se obter um maior conhecimento relativo à eletrofisiologia cardíaca. Nesses experimentos são coletados dados eletrofisiológicos dos domínios intra-, extra- e intercelular de regiões funcionais específicas e do coração como um todo. Os dados são, por exemplo, tensões entre

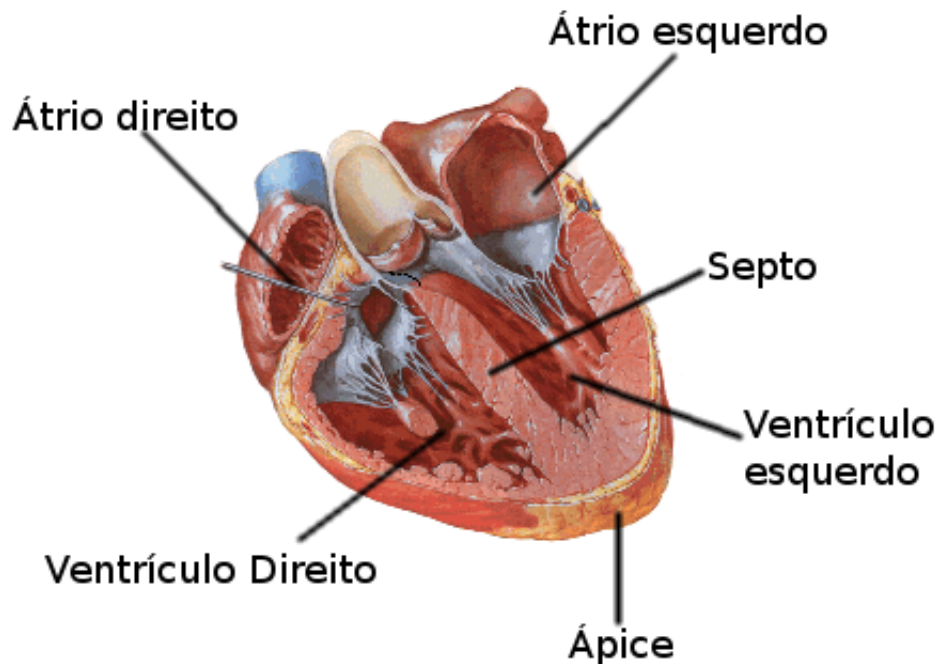


Figura 2.1. Anatomia do coração (adaptado de Netter [2008]).

membranas em diferentes domínios espaciais, fluxos e concentrações de íons. Os experimentos vão de medidas de abertura de um único canal iônico até registros de eletrocardiogramas. Os fenômenos descobertos por esses experimentos são atribuídos, por exemplo, a mudanças do estado eletrofisiológico de componentes celulares (membrana celular, canais iônicos, bombas e estruturas intracelulares como o retículo sarcoplasmático).

Parte dos dados obtidos por esses experimentos é usada para o desenvolvimento de modelos matemáticos com diferentes níveis de abstração. Esses modelos permitem a simulação computacional do comportamento eletrofisiológico pelo uso de métodos numéricos. A reprodução de dados previamente medidos e a descoberta e compreensão de novos fenômenos são alguns dos objetivos da modelagem computacional.

Neste capítulo iremos abordar a modelagem dos diferentes componentes celulares e também do tecido cardíaco, bem como métodos matemáticos para a resolução dos mesmos. No apêndice A, o trabalho clássico de Hodgkin e Huxley [Hodgkin & Huxley, 1952] também é discutido. Nele foram usados dados eletrofisiológicos do axônio gigante da lula para o desenvolvimento de um modelo matemático. A maioria dos modelos matemáticos modernos para células nervosas e musculares é ainda baseada na metodologia desenvolvida por Hodgkin e Huxley. Pelo desenvolvimento desta metodo-

logia Sir John Carew Eccles, Alan Lloyd Hodgkin e Andrew Fielding Huxley receberam o prêmio Nobel de Medicina/Fisiologia em 1963.

2.2 Fisiologia da Membrana Celular

A membrana celular (ou plasmática) engloba a célula, definindo seus limites e separando o meio intracelular do extracelular. A membrana celular é a principal responsável pelo controle de saída e entrada de substâncias da célula, assim como pela manutenção da concentração iônica intracelular distinta da do meio extracelular. Ela é constituída por duas camadas fosfolipídicas fluidas e contínuas, como mostrado na figura 2.2.

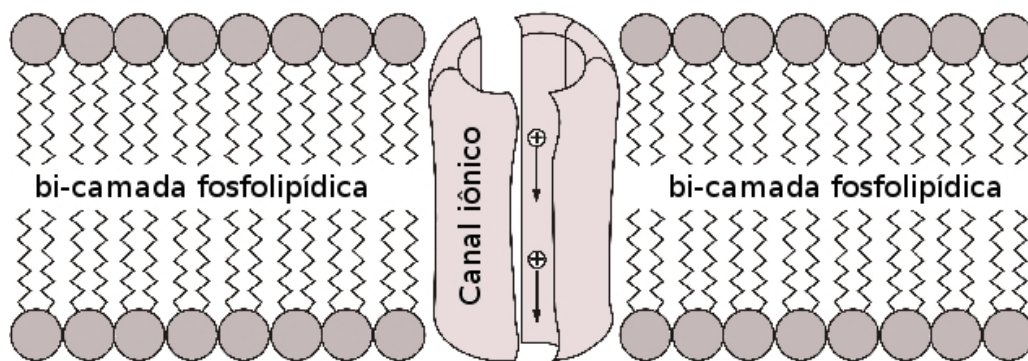


Figura 2.2. Visão da membrana celular (bicamada fosfolipídica e proteínas que a permeiam).

Existem na membrana proteínas com arranjos especiais que formam canais que a permeiam permitindo a passagem de moléculas polares e íons, os canais iônicos. A figura 2.2 mostra uma visão esquemática de como as proteínas formam canais pelos quais íons podem passar. Os canais são especializados e somente uma substância ou um pequeno grupo de íons pode passar através de um canal em particular.

A diferença da composição química e elétrica nos fluídos intra e extracelular gera uma diferença de potencial na membrana, o potencial transmembrânico. Esta diferença de potencial tem papel fundamental na comunicação intercelular, os impulsos elétricos, como veremos a seguir.

2.3 Potencial de Ação e Excitabilidade

Vimos na seção anterior que existe na membrana celular uma diferença de potencial que gera fluxos iônicos através da membrana. A regulação do potencial transmembrânico pelos canais iônicos é uma das funções mais importantes da célula. Vários tipos de células, como neurônios e células musculares, usam este potencial como um sinal. Assim, o funcionamento do sistema nervoso e da contração muscular, por exemplo, dependem da geração e propagação de sinais elétricos, isto é, do potencial de ação.

Para entendermos os sinais elétricos nas células podemos dividir todos os tipos de célula em dois grupos: células excitáveis e células não-excitáveis. Muitas células mantêm um potencial de equilíbrio estável. Para algumas delas, se correntes elétricas são aplicadas em um período curto de tempo, o potencial retorna diretamente para o equilíbrio depois que a corrente é removida. Tais células são chamadas de não-excitáveis.

Entretanto, existem células nas quais uma injeção de corrente suficientemente forte faz com que o potencial transmembrânico percorra um longo caminho, denominado potencial de ação, antes de retornar ao repouso. Tais células são denominadas excitáveis. Podemos citar como células excitáveis: células cardíacas, musculares, secretoras e a maioria dos neurônios. A vantagem mais óbvia da excitabilidade é que uma célula excitável ou responde completamente a um estímulo ou não responde. Assim, um estímulo com amplitude suficiente pode ser distinguido de um simples ruído. Desta maneira, o ruído é filtrado e o sinal é transmitido confiavelmente.

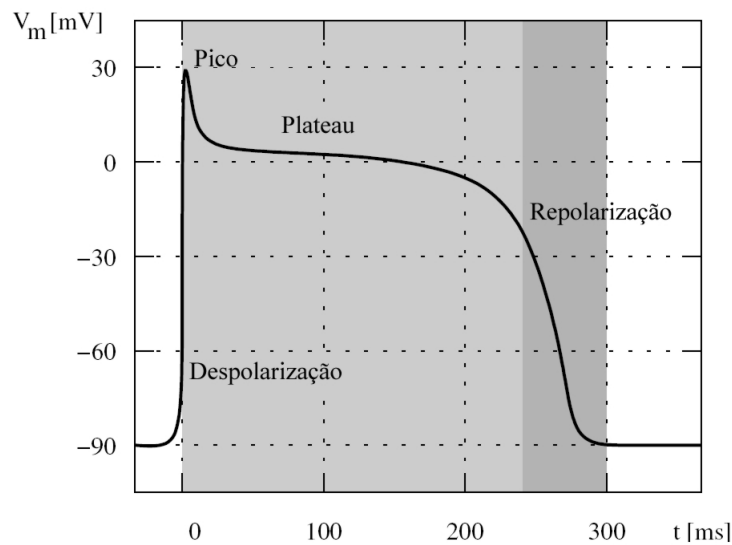


Figura 2.3. Ilustração esquemática de um potencial de ação medido na membrana de um miócito cardíaco (adaptado de [Sachse, 2004]).

A figura 2.3 mostra uma ilustração esquemática de um potencial de ação medido na membrana de um miócito cardíaco. Uma corrente de estímulo é aplicada em um miócito cujo potencial transmembrânico está em repouso. Após uma rápida despolarização, o potencial transmembrânico atinge valores positivos. Depois de uma queda rápida, a fase de *plateau* relativamente longa é iniciada. No final a repolarização leva a célula novamente ao potencial de repouso.

2.4 Modelo Matemático para a Membrana Celular

Se desconsiderarmos a existência de canais iônicos, a principal característica da membrana é a separação de cargas entre o meio extracelular e o meio intracelular. Por esse motivo, a membrana pode ser vista como um capacitor [Hille, 2001]. Sendo assim, o potencial na membrana V_m é proporcional à carga Q :

$$V_m = \frac{Q}{C_m}, \quad (2.1)$$

onde C_m é a capacitância da membrana.

No entanto, a existência de canais iônicos que permitem a passagem de íons pela membrana faz com que ela não possa ser vista como um simples capacitor. É mais correto acrescentar ao modelo um resistor ou elemento não linear acoplado em paralelo ao capacitor, o qual modela a passagem da corrente I_{ion} pelos canais iônicos. A figura 2.4 ilustra a membrana celular e sua aproximação por um circuito resistor-capacitor. O circuito consiste de um resistor não linear R_m e um capacitor C_m . O potencial sobre a membrana V_m é definido pela diferença entre o potencial extracelular ϕ_e e o potencial intracelular ϕ_i .

O fluxo iônico mudará a quantidade de carga separada pela membrana e também o potencial transmembrânico. Podemos calcular a corrente capacitiva I_C da seguinte maneira:

$$\frac{dV_m}{dt} = \frac{d}{dt} \left(\frac{Q}{C_m} \right) = \frac{I_C}{C_m} \quad (2.2)$$

assumindo que a capacitância C_m é constante ao longo do tempo. Essa equação é a base da maioria dos modelos eletrofisiológicos de membranas e células [Sachse, 2004].

A corrente transmembrânica total é a soma das correntes capacitiva e iônica,

$$I_m = I_{ion} + I_C \quad (2.3)$$

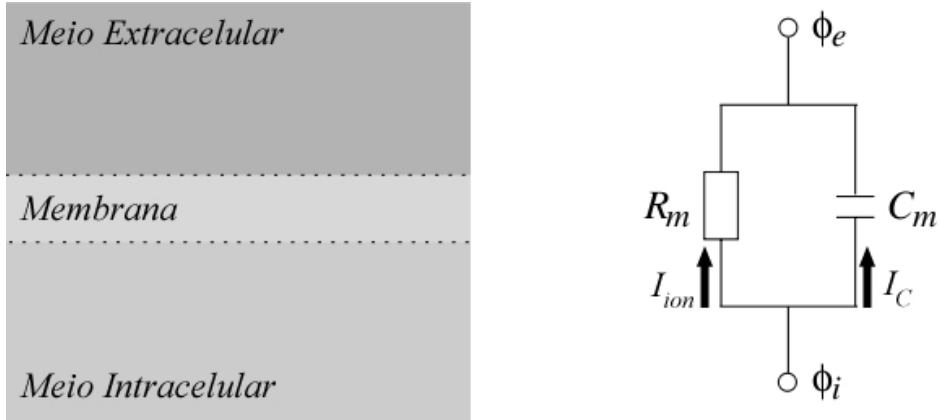


Figura 2.4. A Membrana celular e sua aproximação por um circuito resistor capacitor.

Combinando 2.2 com 2.3 temos:

$$I_m = I_{ion} + C_m \frac{dV_m}{dt} \quad (2.4)$$

Se o circuito na figura 2.4 for fechado, não haverá transporte de íons na malha, então pela conservação da corrente:

$$I_{ion}(\boldsymbol{\eta}, V_m) + C_m \frac{dV_m}{dt} = 0 \quad (2.5)$$

A corrente iônica é normalmente modelada por um sistema de equações diferenciais ordinárias, podendo ter dezenas de variáveis. Dessa forma, podemos escrever um modelo celular da seguinte maneira:

$$C_m \frac{dV_m}{dt} = -I_{ion}(\boldsymbol{\eta}, V_m) \quad (2.6)$$

$$\frac{d\boldsymbol{\eta}}{dt} = f(V_m, \boldsymbol{\eta}) \quad (2.7)$$

Maiores informações sobre a modelagem matemática de I_{ion} podem ser encontradas no apêndice A.

2.5 Modelos para o tecido

As células do coração conectam-se por junções do tipo *gap*, possibilitando a comunicação entre células imediatamente vizinhas e permitindo o fluxo de corrente elétrica na

forma de íons. Deste modo, uma célula estimulada pode transmitir um sinal elétrico para suas vizinhas propagando um estímulo elétrico para outras partes do coração e ativando a contração do órgão.

Nesta seção descrevemos como as células se interconectam formando o tecido cardíaco pelo qual a onda elétrica se propaga e também discutiremos as propriedades condutivas da estrutura do tecido. Para isso dois modelos são apresentados, o monodomínio e o bidomínio, que diferem quanto ao número de domínios utilizados para o fluxo de corrente elétrica.

2.5.1 O modelo bidomínio

O modelo bidomínio é um modelo matemático que captura as propriedades elétricas do músculo cardíaco e descreve a propagação elétrica no tecido. Por ser um modelo contínuo suas equações descrevem o comportamento elétrico de forma macroscópica, ou seja, as muitas células que compõem o tecido são tratadas de forma homogênea, o que é bastante adequado para a maioria das situações [Keener & Sneyd, 1998]. Neste modelo, o tecido é dividido em dois domínios: intracelular e extracelular. Os domínios são considerados contínuos pois, no intracelular as células estão interconectadas através das junções do tipo *gap*, enquanto no extracelular a corrente elétrica flui nos espaços entre as células.

Com base neste modelo, podemos considerar o tecido como um meio onde cada ponto do espaço é composto por uma fração do espaço intracelular e uma fração do espaço extracelular. Dessa forma, os pontos também possuem dois potenciais elétricos denominados V_e e V_i além de duas correntes I_e e I_i , onde os subscritos e e i denotam os espaços extra e intracelular, respectivamente.

A relação entre os potenciais e as correntes pode ser definida pela Lei de Ohm:

$$I_i = -\sigma_i \nabla V_i \quad (2.8)$$

$$I_e = -\sigma_e \nabla V_e \quad (2.9)$$

onde σ_i e σ_e são denominados tensores de condutividade. Em qualquer ponto no espaço a corrente total é $I_t = I_i + I_e$ e, a não ser que existam fontes externas de corrente, a corrente total é conservada, resultando na seguinte equação:

$$\nabla \cdot I_t = \nabla \cdot (\sigma_i \nabla V_i + \sigma_e \nabla V_e) = 0 \quad (2.10)$$

Em cada ponto do espaço existe um potencial transmembrânico V_m

$$V_m = V_i - V_e. \quad (2.11)$$

Existe também uma corrente transmembrânica, que é a corrente que deixa o espaço intracelular e flui para espaço extracelular:

$$I_m = \nabla \cdot (\sigma_i \nabla V_i) = -\nabla \cdot (\sigma_e \nabla V_e) \quad (2.12)$$

Como vimos na seção 2.4, a corrente transmembrânica total é a soma das correntes capacitiva e iônica,

$$I_m = \beta \left(C_m \frac{\partial V_m}{\partial t} + I_{ion} \right) = \nabla \cdot (\sigma_i \nabla V_i) \quad (2.13)$$

onde β é uma constante utilizada para converter a corrente transmembrânica de unidade de área para unidade de volume. Essa constante é necessária pois tanto a corrente iônica I_{ion} como a capacitância C_m são convenientemente medidas por unidade de área da membrana.

As equações 2.10 e 2.13 descrevem o modelo bidomínio completo. O modelo bidomínio foi proposto inicialmente por Tung [1978] e Miller & Geselowitz [1978] e é hoje o modelo aceito para modelar o comportamento da atividade elétrica no tecido cardíaco [Henriquez, 1993].

Se utilizarmos a relação $V_i = V_m + V_e$ (equação (2.11)) e rearranjarmos as equações 2.10 e 2.13 obtemos:

$$\nabla \cdot (\sigma_i \nabla V_m) + \nabla \cdot (\sigma_i \nabla V_e) = \beta \left(C_m \frac{\partial V_m}{\partial t} + I_{ion} \right) \quad (2.14)$$

$$\nabla \cdot (\sigma_i \nabla V_m) + \nabla \cdot ((\sigma_i + \sigma_e) \nabla V_e) = 0 \text{ em } \Omega \quad (2.15)$$

que é a formulação padrão do modelo bidomínio, proposta pela primeira vez por Tung [1978] e Miller & Geselowitz [1978]. Nessa formulação temos que $\sigma_i \nabla V_m \times \vec{\eta} = 0$ e $\sigma_e \nabla V_e \times \vec{\eta} = 0$ em $\partial\Omega$ e $V(t=0, x) = V_0(x)$ em Ω .

É importante citar que o tecido cardíaco é anisotrópico. Sendo assim, a condutividade elétrica não é a mesma em todas as direções o que faz com que σ_i e σ_e sejam representados como tensores de segunda ordem.

Essa diferença nas condutividades ocorre porque o músculo do coração é constituído de fibras, nas quais a condutividade é maior no sentido das fibras. Como as fibras são organizadas em folhas de fibras tem-se três direções para a condutividade: paralela

às fibras, perpendicular às fibras mas paralelo às folhas das fibras ou perpendicular as folhas. Além disso, a taxa de anisotropia é diferente nos meios intra e extracelulares [Roth, 1997]. Maiores informações sobre a condutividade no tecido cardíaco podem ser encontradas em Sundnes [2006].

2.5.2 O modelo monodomínio

O modelo bidomínio pode ser simplificado se levarmos em consideração que as condutividades intra e extracelulares são proporcionais e satisfazem a seguinte regulação [Sundnes et al., 2006]:

$$\sigma_e = \lambda \sigma_i \quad (2.16)$$

onde λ é uma constante.

Com isso, podemos simplificar as equações 2.14 e 2.15:

$$\nabla \cdot (\sigma_i \nabla V_m) + \nabla \cdot (\sigma_i \nabla V_e) = \beta \left(C_m \frac{\partial V_m}{\partial t} + I_{ion} \right) \quad (2.17)$$

$$\nabla \cdot (\sigma_i \nabla V_m) + (1 + \lambda) \nabla \cdot (\sigma_i \nabla V_e) = 0 \quad (2.18)$$

Podemos também reescrever a equação (2.18)

$$\nabla \cdot (\sigma_i \nabla V_e) = -\frac{1}{(1 + \lambda)} \nabla \cdot (\sigma_i \nabla V_m) \quad (2.19)$$

e substituí-la na equação (2.17) obtendo:

$$\frac{\lambda}{(1 + \lambda)} \nabla \cdot (\sigma_i \nabla V_m) = \beta \left(C_m \frac{\partial V_m}{\partial t} + I_{ion} \right) \quad (2.20)$$

Fazendo $\sigma = \frac{\sigma_i \lambda}{(1 + \lambda)}$ temos:

$$\nabla \sigma \nabla V_m = \beta \left(C_m \frac{\partial V_m}{\partial t} + I_{ion} \right) \quad (2.21)$$

onde $\sigma_i \nabla V_m \times \vec{\eta} = 0$ em $\partial\Omega$ e $V(t = 0, x) = V_0(x)$ em Ω .

Cabe ressaltar que o modelo monodomínio é mais simples tanto matematicamente quanto computacionalmente. Além disso, por considerar taxas de condutividade proporcionais para todos os meios, o monodomínio apresenta algumas limitações, tornando-se útil para análises e estudos mais simples. Simulações mais realistas devem utilizar o modelo bidomínio.

É importante observar também que o termo I_{ion} presente nas equações anteriores depende dos modelos celulares utilizados. Exemplos desse modelos e do cálculo de I_{ion} são apresentados nas seções A.4, A.5 e A.6 do apêndice A.

2.6 Discretização temporal e operador *splitting*

Nas seções anteriores apresentamos as equações do bidomínio (eq. 2.15) e monodomínio (eq. 2.20) que modelam a atividade elétrica no coração. Neste ponto, essas equações podem ser consideradas como um conjunto acoplado de equações diferenciais parciais (EDPs) elípticas e parabólicas e equações diferenciais ordinárias (EDOs), no caso do bidomínio, ou somente EDPs parabólicas e EDOs no monodomínio. Devido à natureza altamente não-linear das EDOs, soluções utilizando métodos totalmente implícitos são extremamente difíceis [Vigmond et al., 2008]. Além disso as EDOs que descrevem a dinâmica das membranas celulares estão se tornando cada vez mais complicadas, podendo exigir mais de 100 equações com mais de 60 equações diferenciais acopladas [Iyer et al., 2004].

Para resolvermos o problema da solução das EDOs podemos lançar mão de uma técnica denominada operador *splitting* [Strikwerda, 2004]. O operador *splitting* é uma técnica para resolver uma expressão complexa como uma sequência de expressões simples. Isso nos permite aplicar diferentes métodos numéricos para a resolução de cada passo da sequência com o intuito de aumentar a eficiência computacional e também eliminar dependências complexas entre as variáveis. A desvantagem do uso dessa técnica é uma possível perda na precisão da solução, pois as dependências entre as variáveis não são satisfeitas simultaneamente.

No caso das equações dos modelos bidomínio e monodomínio, utilizamos o operador *splitting* para separarmos as EDOs das EDPs. Essa separação permite, por exemplo, o uso de métodos explícitos para a resolução das EDOs (por exemplo: Método de Euler Explícito [LeVeque, 2007]) e métodos implícitos (por exemplo: Método de Crank–Nicolson [Crank & Nicolson, 1947]) para a solução das EDPs.

Para ilustrarmos o uso do operador *splitting* iremos reescrever as equações do bidomínio 2.14 e 2.15 da seguinte maneira:

$$\nabla \cdot (\sigma_i \nabla) + \nabla \cdot (\sigma_i \nabla V_e) = \beta \left(C_m \frac{\partial V}{\partial t} + I_{ion}(V, \boldsymbol{\eta}) \right) \quad (2.22)$$

$$\nabla \cdot (\sigma_i \nabla V) + \nabla \cdot ((\sigma_i + \sigma_e) \nabla V_e) = 0 \quad (2.23)$$

$$\frac{\partial \boldsymbol{\eta}}{\partial t} = f(V, \boldsymbol{\eta}) \quad (2.24)$$

Após a aplicação do operador *splitting* a solução numérica se reduz a um esquema de 3 passos que envolve a solução de uma EDP parabólica, uma EDP elíptica e um sistema não linear de EDOs a cada passo de tempo. Reescrevendo as equações (2.22)-(2.24) usando uma técnica de operador *splitting* (veja [Weber dos Santos et al., 2004b] para mais detalhes) e um método implícito nós obtemos o seguinte esquema numérico:

$$v^{k+1/2} = (1 + \Delta t A_i) v^{k+1/2} + \Delta t A_i (v_e^k) \quad (2.25)$$

$$v^{k+1} = v^{k+1/2} - \frac{\Delta t I_{ion}(v^{k+1/2}, \vec{\xi}^k)}{C_m} \quad (2.26)$$

$$\vec{\xi}^{k+1} = \vec{\xi}^k + \Delta t g(v^{k+1/2}, \vec{\xi}^k) \quad (2.27)$$

$$(A_i + A_e)(v_e)^{k+1} = -A_i v^{k+1} \quad (2.28)$$

onde v^k, v_e^k e $\vec{\xi}^k$ são as discretizações de V_m, V_e e $\vec{n}$ no tempo $k \cdot \Delta t$; A_i e A_e são as discretizações no espaço para $\nabla \cdot (\sigma_i \nabla)$ e $\nabla \cdot (\sigma_e \nabla)$, respectivamente. A discretização espacial pode ser feita utilizando vários métodos.

Já para o monodomínio temos:

$$v^{k+1/2} = (1 + \Delta t A) v^{k+1/2} \quad (2.29)$$

$$v^{k+1} = v^{k+1/2} - \frac{\Delta t I_{ion}(v^{k+1/2}, \vec{\xi}^k)}{C_m} \quad (2.30)$$

$$\vec{\xi}^{k+1} = \vec{\xi}^k + \Delta t g(v^{k+1/2}, \vec{\xi}^k) \quad (2.31)$$

onde v^k e $\vec{\xi}^k$ são as discretizações de V_m e η no tempo $k \cdot \Delta t$ e A é a discretizações no espaço para $\nabla \cdot (\sigma \nabla)$.

2.7 O método dos volumes finitos

O método dos volumes finitos (MVF) é um método matemático utilizado para se obter uma versão discreta de equações diferenciais parciais (EDPs). Este método é adequado para simulações numéricas de vários tipos de leis de conservação (elíptica, parabólica ou hiperbólica) [Eymard et al., 2000]. Assim como o método dos elementos finitos (MEF), o MVF pode ser usado em vários tipos de geometria, usando malhas estruturadas ou não estruturadas e gera esquemas numéricos robustos [Eymard et al., 2000]. Além disso, o MVF tem sido extensamente utilizado em problemas envolvendo leis de conservação, como por exemplo simulação de reservatórios na engenharia de petróleo, por ser um método conservativo, no sentido de conservar numericamente os fluxos de uma célula para a sua vizinha.

O desenvolvimento do método está intrinsecamente ligado ao conceito de fluxo entre regiões, ou volumes adjacentes, onde o fluxo de uma grandeza, como massa ou energia, é a quantidade dessa grandeza φ que atravessa uma fronteira com área A . A quantidade de φ que atravessa um volume de controle Ω_c por unidade de tempo é calculada pela integração, sobre essas fronteiras, da diferença entre os fluxos que entram e os que saem de Ω_c , o que é conseguido de forma mais geral pela integração das EDPs [Bortoli, 2000]. Para alguns problemas isotrópicos discretizados com malhas regulares, a discretização obtida com o MVF é muito parecida com aquela obtida pela aplicação do método de diferenças finitas (MDF).

2.7.1 O MVF aplicado ao modelo monodomínio

Nesta seção faremos uma breve descrição da aplicação do MVF à discretização das equações do modelo monodomínio. Para isso, utilizaremos a equação (2.20), descrita na seção 2.5.2. Informações mais detalhadas sobre o MVF aplicado ao monodomínio podem ser encontradas em Harrild & Henriquez [1997] e Coudiere et al. [2009].

Por questões de legibilidade iremos repetir uma versão simplificada da equação (2.21) e iremos considerá-la para futuras referências.

$$\beta \left(C_m \frac{\partial V}{\partial t} + I_{ion} \right) = \nabla \cdot (\sigma \nabla V) \quad (2.32)$$

2.7.1.1 Discretização no tempo

A derivada no tempo presente na equação (2.32), que opera sobre grandeza V , será aproximada por um esquema de Euler implícito de primeira ordem:

$$\frac{\partial V}{\partial t} = \frac{V^{n+1} - V^n}{\Delta t_p}, \quad (2.33)$$

onde V^n representa o potencial transmembrânico no instante de tempo t_n e Δt_p é o passo de tempo utilizado para avançar a EDP no tempo. Como o sistema não-linear de EDOs é do tipo *stiff* [Sundnes et al., 2003b], a sua discretização exige passos de tempo muito pequenos. Para os modelos simples baseados na formulação de Hodgkin-Huxley, este problema é normalmente contornado com a utilização do método de Rush-Larsen (RL)[Rush & Larsen, 1978]. No entanto, para os modelos mais modernos e complexos que são altamente baseados em Cadeias de Markov (CM), o método de RL parece ser ineficaz em termos de permitir passos de tempo maiores durante a integração numérica. Para o caso modelo de Bondarenko et al. [2004], testou-se ambos os métodos, Euler e RL, e ambos exigiram o mesmo passo $\Delta t_e = 0,0001 \text{ ms}$ por questões de estabilidade,

onde Δt_e é o passo de tempo utilizado para avançar as EDOs no tempo. Como o método de RL é mais caro por passo de tempo do que o método de Euler, neste trabalho foi utilizado o método simples de Euler explícito para a discretização das EDOs não-lineares.

No entanto, como já dissemos acima, podemos utilizar passos de tempo diferentes para a solução dos dois diferentes problemas desacoplados, a EDP e as EDOs. Uma vez que utilizamos um método incondicionalmente estável para a EDP (Euler implícito), o passo de tempo Δt_p pode ser muito maior do que o utilizado para a solução do sistema linear de EDOs, $\Delta t_o = 0,0001 \text{ ms}$. Neste trabalho, testamos valores diferentes de Δt_p , até mesmo alguns valores maiores que o valor de Δt_o .

2.7.1.2 Discretização no espaço

O termo difusivo da equação (2.32) precisa ser discretizado espacialmente. Para isso iremos considerar as seguintes relações:

$$J = -\sigma \nabla V \quad (2.34)$$

onde J ($\mu A/cm^2$) expressa a densidade do fluxo de corrente intracelular e

$$\nabla \cdot J = -I_v. \quad (2.35)$$

Nessa expressão, I_v ($\mu A/cm^3$) é uma corrente volumétrica e corresponde ao lado esquerdo da equação (2.32), servindo de base para a solução baseada em volumes finitos.

Para a discretização no espaço iremos considerar uma malha uniforme, bidimensional, formada por quadriláteros regulares (denominados “volumes”). Situado no centro de cada volume se encontra um nó. A quantidade de interesse V é associada a cada nó da malha.

Após a definição da geometria da malha, as equações específicas do MVF podem ser apresentadas. A equação (2.35) pode ser integrada espacialmente sobre um quadrilátero específico, levando a:

$$\int_{\Omega} \nabla \cdot J dv = - \int_{\Omega} I_v dv. \quad (2.36)$$

Aplicando o teorema da divergência temos,

$$\int_{\Omega} \nabla \cdot J dv = \int_{\partial\Omega} J \cdot \vec{\xi} ds, \quad (2.37)$$

onde $\vec{\xi}$ representa o vetor unitário normal à fronteira $\partial\Omega$. Com isso podemos reescrever a equação (2.34) como

$$\int_{\partial\Omega} J \cdot \vec{\xi} = - \int_{\Omega} I_v ds. \quad (2.38)$$

Finalmente, assumindo que I_v representa um valor médio em cada quadrilátero em particular e substituindo na equação (2.32), temos a seguinte relação:

$$\beta \left(C_m \frac{\partial V}{\partial t} \right) \Big|_{(i,j)} = \frac{- \int_{\partial\Omega} J_{i,j} \cdot \vec{\xi} ds}{A_{i,j}}, \quad (2.39)$$

onde $A_{i,j}$ é a área do volume de controle. Note que a equação (2.39) pode ser reescrita em 3 dimensões simplesmente substituindo a integral no contorno por uma integral de área sobre a superfície de um volume hexaédrico e dividindo pelo volume ao invés da área do elemento. Por questões de simplicidade iremos levar em consideração somente o caso bidimensional.

Para este problema bidimensional em particular, formado por uma malha uniforme de quadriláteros de lado h , o cálculo de $J_{i,j}$ pode ser subdividido como a soma dos fluxos nas faces:

$$\int_{\partial\Omega} J_{i,j} \cdot \vec{\xi} = (J_{x_{i+1/2,j}} - J_{x_{i-1/2,j}} + J_{y_{i,j+1/2}} - J_{y_{i,j-1/2}})h \quad (2.40)$$

onde $J_{x_{m,n}}$ e $J_{y_{m,n}}$ são calculados nas faces $((m,n) = (i+1/2, j), (i-1/2, j), (i, j+1/2),$ ou $(i, j-1/2))$ da seguinte forma:

$$J_{x_{m,n}} = \sigma_x(m,n) \frac{\partial V}{\partial x} \Big|_{(m,n)} + \sigma_{xy}(m,n) \frac{\partial V}{\partial y} \Big|_{(m,n)} \quad (2.41)$$

$$J_{y_{m,n}} = \sigma_{xy}(m,n) \frac{\partial V}{\partial x} \Big|_{(m,n)} + \sigma_y(m,n) \frac{\partial V}{\partial y} \Big|_{(m,n)} \quad (2.42)$$

O tensor $\sigma = \begin{pmatrix} \sigma_x & \sigma_{xy} \\ \sigma_{xy} & \sigma_y \end{pmatrix}$ deve ser avaliado nas interfaces do volume. Para isso, utilizamos a média harmônica:

$$\sigma_{x_{i+1/2,j}} = \frac{2\sigma_{x_{i,j}}\sigma_{x_{i+1,j}}}{\sigma_{x_{i+1,j}} + \sigma_{x_{i,j}}} \quad (2.43)$$

$$\sigma_{x_{i-1/2,j}} = \frac{2\sigma_{x_{i,j}}\sigma_{x_{i-1,j}}}{\sigma_{x_{i-1,j}} + \sigma_{x_{i,j}}} \quad (2.44)$$

podemos usar um raciocínio análogo para σ_y . Já para σ_{xy} temos que dividi-lo em dois

fatores, que serão utilizados quando as equações A.51, A.52, A.55 e A.56, descritas no apêndice A, forem utilizadas:

$$\sigma_{xy_{i+1/2},j}^1 = \frac{2\sigma_{xy_{i+1},j} \cdot \sigma_{x_{i,j}}}{\sigma_{x_{i+1},j} + \sigma_{x_{i,j}}} \quad (2.45)$$

$$\sigma_{xy_{i+1/2},j}^2 = \frac{2\sigma_{xy_{i,j}} \cdot \sigma_{x_{i+1},j}}{\sigma_{x_{i+1},j} + \sigma_{x_{i,j}}} \quad (2.46)$$

$$\sigma_{xy_{i,j+1/2}}^1 = \frac{2\sigma_{xy_{i,j+1}} \cdot \sigma_{y_{i,j}}}{\sigma_{y_{i,j+1}} + \sigma_{y_{i,j}}} \quad (2.47)$$

$$\sigma_{xy_{i,j+1/2}}^2 = \frac{2\sigma_{xy_{i,j}} \cdot \sigma_{y_{i,j+1}}}{\sigma_{y_{i,j+1}} + \sigma_{y_{i,j}}} \quad (2.48)$$

$$\sigma_{xy_{i-1/2},j}^1 = \frac{2\sigma_{xy_{i-1},j} \cdot \sigma_{x_{i,j}}}{\sigma_{x_{i-1},j} + \sigma_{x_{i,j}}} \quad (2.49)$$

$$\sigma_{xy_{i-1/2},j}^2 = \frac{2\sigma_{xy_{i,j}} \cdot \sigma_{x_{i-1},j}}{\sigma_{x_{i-1},j} + \sigma_{x_{i,j}}} \quad (2.50)$$

$$\sigma_{xy_{i,j-1/2}}^1 = \frac{2\sigma_{xy_{i,j-1}} \cdot \sigma_{y_{i,j}}}{\sigma_{y_{i,j-1}} + \sigma_{y_{i,j}}} \quad (2.51)$$

$$\sigma_{xy_{i,j-1/2}}^2 = \frac{2\sigma_{xy_{i,j}} \cdot \sigma_{y_{i,j-1}}}{\sigma_{y_{i,j-1}} + \sigma_{y_{i,j}}} \quad (2.52)$$

2.7.1.3 Malha uniforme não-adaptativa

Nesta seção iremos mostrar a aplicação do MVF se utilizarmos uma malha uniforme e não-adaptativa. As derivadas parciais de V nas interfaces serão aproximadas usando um esquema clássico de diferenças finitas, considerando discretizações uniformes no espaço ($\Delta x = \Delta y = h$).

Desenvolvendo todas as equações necessárias podemos definir a fórmula de avanço

no tempo para os pontos interiores da malha:

$$\begin{aligned}
& (\sigma_{x_{i+1/2,j}} + \sigma_{x_{i-1/2,j}} + \sigma_{y_{i,j+1/2}} + \sigma_{y_{i,j-1/2}} + \alpha) V_{m_{i,j}}^* + \\
& \quad \frac{-\sigma_{xy_{i-1/2,j}}^1 - \sigma_{xy_{i,j-1/2}}^1}{4} V_{m_{i-1,j-1}}^* + \\
& \quad \frac{\sigma_{xy_{i+1/2,j}}^2 - \sigma_{xy_{i-1/2,j}}^2 - 4\sigma_{y_{i,j-1/2}}}{4} V_{m_{i,j-1}}^* + \\
& \quad \frac{\sigma_{xy_{i+1/2,j}}^1 + \sigma_{xy_{i,j-1/2}}^1}{4} V_{m_{i+1,j-1}}^* + \\
& \quad \frac{-4\sigma_{x_{i+1/2,j}} - \sigma_{xy_{i,j+1/2}}^2 + \sigma_{xy_{i,j-1/2}}^2}{4} V_{m_{i+1,j}}^* + \\
& \quad \frac{-\sigma_{xy_{i+1/2,j}}^1 - \sigma_{xy_{i,j+1/2}}^1}{4} V_{m_{i+1,j+1}}^* + \\
& \quad \frac{-\sigma_{xy_{i+1/2,j}}^2 + \sigma_{xy_{i-1/2,j}}^2 - 4\sigma_{y_{i,j+1/2}}}{4} V_{m_{i,j+1}}^* + \\
& \quad \frac{\sigma_{xy_{i-1/2,j}}^1 + \sigma_{xy_{i,j+1/2}}^1}{4} V_{m_{i-1,j+1}}^* + \\
& \quad \frac{-4\sigma_{x_{i-1/2,j}} + \sigma_{xy_{i,j+1/2}}^2 - \sigma_{xy_{i,j-1/2}}^2}{4} V_{m_{i-1,j}}^* = V_{m_{i,j}}^n \cdot \alpha,
\end{aligned} \tag{2.53}$$

onde $\alpha = \frac{\beta C_m h^2}{\Delta t}$. Detalhes sobre o desenvolvimento das equações podem ser encontrados no apêndice A.

Simplificação para fibra horizontal ($\sigma_{xy} = 0$) Em um tecido com as fibras orientadas na direção horizontal, a condutividade $\sigma_{xy} = 0$. Com isso, as equações que definem os fluxos (A.62, A.63, A.64 e A.66) podem ser simplificadas da seguinte forma maneira:

$$J_{x_{i+1/2,j}} = \sigma_{x_{i+1/2,j}} \frac{V_{m_{i+1,j}} - V_{m_{i,j}}}{h} \tag{2.54}$$

$$J_{x_{i-1/2,j}} = \sigma_{x_{i-1/2,j}} \frac{V_{m_{i,j}} - V_{m_{i-1,j}}}{h} \tag{2.55}$$

$$J_{y_{i,j+1/2}} = \sigma_{y_{i,j+1/2}} \frac{V_{m_{i,j+1}} - V_{m_{i,j}}}{h} \tag{2.56}$$

$$J_{y_{i,j-1/2}} = \sigma_{y_{i,j-1/2}} \frac{V_{m_{i,j}} - V_{m_{i,j-1}}}{h} \tag{2.57}$$

Desenvolvendo todas as equações, podemos agora definir a fórmula de avanço no

tempo para os pontos interiores da malha:

$$\begin{aligned}
 & (\sigma_{x_{i+1/2,j}} + \sigma_{x_{i-1/2,j}} + \sigma_{y_{i,j+1/2}} + \sigma_{y_{i,j-1/2}} + \alpha) V_{m_{i,j}}^* - \\
 & \quad \sigma_{y_{i,j-1/2}} V_{m_{i,j-1}}^* - \\
 & \quad \sigma_{x_{i+1/2,j}} V_{m_{i+1,j}}^* - \\
 & \quad \sigma_{y_{i,j+1/2}} V_{m_{i,j+1}}^* - \\
 & \quad \sigma_{x_{i-1/2,j}} V_{m_{i-1,j}}^* = V_{m_{i,j}}^t * \alpha,
 \end{aligned} \tag{2.58}$$

onde $\alpha = \frac{\beta C_m h^2}{\Delta t_p}$.

As equações mostradas acima resultam em um sistema de equações lineares que deve ser resolvido a cada passo de tempo da simulação.

2.8 Simulando a eletrofisiologia cardíaca

Para realizarmos uma simulação, utilizando por exemplo o modelo monodomínio, o sistema linear apresentado na equação (2.53) ou equação (2.58), referente à discretização da equação parabólica, deve ser resolvido repetidamente, dependendo do tempo de simulação desejado e do Δt empregado. Além disso, a cada passo de tempo, um sistema não-linear de equações ordinárias, proveniente do modelo celular utilizado, deve ser resolvido. Para o caso do modelo bidomínio, além do sistema de EDOs, dois sistemas lineares deverão ser resolvidos, um referente à discretização da equação parabólica e outro da elíptica.

Vários métodos computacionais de resolução de sistemas lineares podem ser utilizados para a realização das simulações. Um método bastante empregado é o Gradiente Conjugado (GC). O GC é um algoritmo iterativo para a solução numérica de sistemas de equações lineares cuja matriz é simétrica e positiva definida, proposto originalmente por Hestenes & Stiefel [1952]. O uso de preconditionadores para o GC, com o intuito de acelerar a resolução dos sistemas, também é bastante difundido, como pode ser visto em Street & Plonsey [1999], Vigmond et al. [2002] e Weber dos Santos et al. [2004a]

A figura 2.5 mostra alguns passos de tempos de uma simulação de arritmia cardíaca, caracterizada por uma onda em espiral intermitente. O modelo matemático utilizado para o tecido foi o monodomínio e a simulação em questão foi realizada por 200 ms utilizando um pedaço bidimensional de tecido cardíaco, medindo $1,2\text{ cm} \times 1,2\text{ cm}$ e com discretização espacial de $0,01\text{ cm}$. Essa discretização gerou um sistema linear de 128 equações e 128 incógnitas que foi resolvido 20.000 vezes ($\Delta t = 0,01$) em uma máquina sequencial utilizando o método GC sem preconditionador. Para a

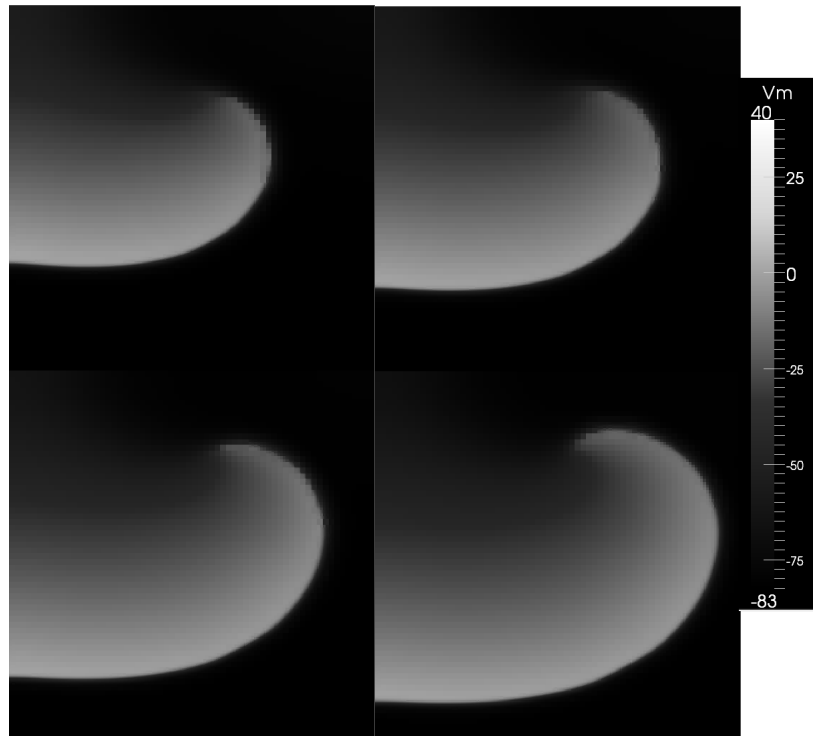


Figura 2.5. Onda espiral gerada pelo modelo monodomínio utilizando volumes finitos e o modelo celular de C. Luo & Yoram Rudy [1991]

dinâmica de I_{ion} utilizamos o modelo celular desenvolvido por C. Luo & Yoram Rudy [1991] e os sistemas de EDOs deste modelo foram integrados utilizando o método de Euler.

Apesar da discretização espacial relativamente grossa ($0,01\text{ cm}$), a simulação foi executada em 29 minutos, ou seja, aproximadamente 8.787 vezes mais lenta que o tempo real. Isso mostra que, se quisermos diminuir os tempos de execução para que sejam pelo menos próximos ao tempo real, devemos acelerar as simulações utilizando técnicas como computação paralela, GPGPU, malhas adaptativas, dentre outras. Algumas técnicas de aceleração serão desenvolvidas e apresentadas nesta tese.

Capítulo 3

Estratégias para aceleração das simulações da eletrofisiologia cardíaca

Neste capítulo iremos descrever nossas propostas de estratégias para a aceleração das simulações da eletrofisiologia cardíaca. Nosso principal objetivo é combinar estratégias de computação paralela, malhas adaptativas no espaço e métodos adaptativos no tempo para aumentar o desempenho das simulações da eletrofisiologia cardíaca.

3.1 Computação paralela

A computação paralela e distribuída é uma tecnologia chave nos dias de hoje, graças a sistemas de alto desempenho interligados por redes rápidas e novas máquinas multiprocessadas. A melhora de desempenho pode ser alcançada, em princípio, adicionando processadores, memória e conexões de rede e colocando todos para trabalharem juntos na resolução de um dado problema [Kumar, 2002]. Dividindo a carga de trabalho, se espera que um sistema com N processadores levará a uma melhora (*speedup*) no tempo de computação. O *speedup* é definido como sendo o tempo de execução de uma determinada instância de um problema em um único processador, dividido pelo tempo de resolução do mesmo problema em uma arquitetura com N processadores. No caso ideal, o *speedup* máximo é igual a N . Porém, existem vários fatores que podem reduzir significativamente o desempenho teórico, como, por exemplo, o tempo de execução da porção sequencial do código, o tempo de comunicação e a sincronização entre os processos [Quinn, 2003].

Neste trabalho propomos a utilização de computação paralela para acelerar as simulações da eletrofisiologia cardíaca. Nossa proposta é utilizar tanto programação em memória compartilhada (CPUs com múltiplos núcleos) quanto GPGPU para aumentar o desempenho das técnicas descritas a seguir.

3.2 Utilização de GPUs

O uso das unidades de processamento gráfico (GPUs) para programação de propósito geral (GPGPU) está se tornando mais popular e há diferentes abordagens para este tipo de programação paralela. Atualmente, o ambiente mais popular para programação é a arquitetura de computação paralela NVIDIA CUDA. Estudos recentes têm relatado ganhos significativos de performance obtidos com NVIDIA CUDA para paralelizar a solução tanto do modelo monodomínio [D. Sato et al., 2009; Rocha et al., 2010] quanto do bidomínio [Amorim & dos Santos, 2012].

Os resultados obtidos por [Bartocci et al., 2011] demonstraram que as GPUs são dispositivos apropriados para as simulações cardíacas. Neste trabalho foi mostrado que a utilização de GPUs permite obter tempos de simulação apenas 50 vezes mais lentos que o tempo real para modelos celulares simplificados e cerca de 18.000 vezes mais lentos para modelos mais complexos.

Nossa proposta é desenvolver e comparar o desempenho de diferentes abordagens de programação GPU para acelerar a resolução do problema do monodomínio. Neste trabalho, apresentaremos uma comparação entre diferentes implementações usando OpenGL, NVIDIA CUDA e OpenCL para a solução numérica das equações monodomínio em plataformas modernas de hardware, afim de identificar a abordagem mais adequada e eficiente para a aceleração das simulações da eletrofisiologia cardíaca.

3.3 Utilização de malhas adaptativas

No momento da propagação da onda elétrica no coração, somente uma fração do meio excitável é ocupado por frentes de onda [Cherry et al., 2000]. Nessas regiões, a solução ou suas derivadas mudam rapidamente. Por esse motivo, a solução numérica das equações diferenciais nessas regiões requer o uso de uma malha computacional extremamente refinada. Sendo assim, o uso de malhas uniformes leva a um custo computacional muito alto, pois devem ter um número muito grande de pontos. Portanto, procedimentos adaptativos que levem em consideração a diferença de escala dos fenômenos apresentam soluções confiáveis e eficientes [Burgarelli et al., 2006].

3.3.1 Trabalhos relacionados

Recentemente, o uso de refinamento adaptativo para obter malhas adequadas à representação eletrofisiologia cardíaca vêm sendo investigado. Um método adaptativo no tempo e espaço utilizando diferenças finitas é apresentado em Cherry et al. [2000]. Este método utiliza partes aninhadas de grades cartesianas uniformes e é guiado por estimadores de erro obtidos pela extrapolação de Richardson. Os resultados apresentados, mostram uma taxa de aceleração de $5\times$, sem perda de precisão.

Elementos *Mortar* são usados em Trangenstein & Kim [2004] para definir diferentes malhas de elementos finitos, em diferentes partes do domínio. O estimador de erro se baseia numa abordagem hierárquica utilizado para controlar tanto a adaptabilidade espacial quanto a temporal. A taxa de aceleração máxima obtida neste trabalho foi de $1,72\times$.

Uma série de publicações recentes [Franzone et al., 2007; Deuffhard et al., 2009; Weiser et al., 2010] descrevem a aplicação da biblioteca Kardos (para a solução de sistemas não lineares de EDPs parabólicas) como método adaptativo de elementos finitos para eletrofisiologia cardíaca. Em Franzone et al. [2007], geometrias simples são consideradas utilizando uma abordagem adaptativa baseada em refinamento hierárquico da malha de acordo com a estimadores de erro *a posteriori*. O nível de refinamento ou o número total de pontos da malha é geralmente limitado. Este trabalho foi estendido com base em Deuffhard et al. [2009], utilizando exemplos realistas de geometrias tridimensionais. Até cinco níveis de refinamento hierárquico (isotrópico), são utilizadas e isso leva a utilização de 2,1 milhões de vértices. Os autores comentam que a simulação de 800 ms em uma máquina AMD com 16 núcleos leva seis semanas de tempo de CPU (o que representa uma degradação no desempenho em comparação com um esquema não-adaptativo). Os autores também utilizam sua experiência anterior com Kardos para discutir os prováveis efeitos da natureza multi-escala do problema e comparam a complexidade associada com a resolução das equações do bidomínio e monodomínio com várias opções de modelo de celular e como isso pode impactar a eficiência de um sistema adaptativo. Mais recentemente, em Weiser et al. [2010] uma discussão interessante é apresentada nos resultados da utilização de métodos adaptativos na simulação cardíaca até o presente momento. Por exemplo, um fator de redução de 150 no número de vértices em comparação com a utilização de malhas fixas com a mesma discretização mais fina foi identificado, mas isso não se traduziu em redução de requisitos de CPU. Algumas das razões para isso são dadas e incluem: limitações passo de tempo pela velocidade da frente e a sua largura, o cálculo da estimativa do erro pode ser uma parte significativa do total da tempo computacional; modificações da malha

levando à necessidade de se re-computar matrizes de discretização, e, finalmente, o custo das modificações da malha em si e as estruturas de dados não-locais associadas também tem o seu preço. Como será mostrado nos resultados deste trabalho, esses problemas não foram detectados com a utilização da malha ALG.

Além disso, temos o trabalho proposto Southern et al. [2012] onde os autores paralelizaram um algoritmo de malha adaptativa desenvolvido em Pain et al. [2001] e obtiveram uma taxa de aceleração máxima de $1024\times$, utilizando um *cluster* com 64 nós. Também podemos citar o trabalho de Southern et al. [2010] onde uma técnica de malha adaptativa anisotrópica é desenvolvida e aplicada a simulações da eletrofisiologia cardíaca, atingindo uma taxa de aceleração de no máximo $12\times$.

Nossa proposta consiste na utilização de uma malha com refinamento adaptativo usando uma abordagem denominada *Autonomous Leaves Graph* (ALG). O ALG é uma estrutura de dados que pode ser integrada ao resolutor de sistemas lineares e representa adequadamente diferentes geometrias, com refinamento adaptativo de fronteiras complexas.

Neste trabalho, decidimos utilizar o ALG pois, diferente dos métodos usados em Bendahmane et al. [2010], Cherry et al. [2003], Franzone et al. [2007] e Southern et al. [2010] ALG é uma estrutura de dados genérica, que não depende de qualquer tipo de método numérico, da geometria do problema ou da natureza do problema. Por exemplo, em Cherry et al. [2003] o método adaptativo foi desenvolvido para diferenças finitas e em Franzone et al. [2007] para elementos finitos. A geometria adotada por Southern et al. [Southern et al., 2010, 2012] depende do método dos elementos finitos e malhas tetraédricas.

A estrutura de dados ALG é flexível o suficiente para lidar com todas essas tarefas ou implementações particulares. Além disso, como descrito em Burgarelli et al. [2006], ALG provou ser computacionalmente eficiente, apresentando custos computacionais mais baixos que outras técnicas normalmente empregadas nesse contexto, como quadtree e octree. O método ALG também foi aplicado para a solução do problema termo-acústico, o qual envolve a solução de um sistema linear proveniente da discretização das equações de Navier–Stokes [Burgarelli & Kischinhevsky, 1999]. Vale ressaltar que a utilização e formulação do método ALG em simulações da eletrofisiologia cardíaca é uma das contribuições deste trabalho, visto que essa abordagem não foi encontrada nas pesquisas bibliográficas realizadas.

3.4 Utilização de passo de tempo adaptativo

Como dito anteriormente, as células cardíacas são usualmente modeladas na forma de um sistema de Equações Diferenciais Ordinárias (EDOs), que são equações que envolvem derivadas das funções com apenas uma variável independente, geralmente a variável t , representando o tempo.

Para resolver numericamente os sistemas de EDOs que compõem os modelos celulares normalmente lançamos mão de métodos numéricos computacionais. Como discutido na seção 2.7.1.1, optamos neste trabalho por utilizar o método Euler explícito. Embora seja conhecido que métodos numéricos explícitos possuem fortes limitações por causa das restrições de estabilidade e precisão [MacLachlan et al., 2007], estes são amplamente utilizados devido à sua simplicidade de implementação [Vigmond et al., 2002; Plank et al., 2007].

Porém, o método de Euler explícito necessita de um passo de tempo muito pequeno para que seu percentual de erro seja aceitável ou até mesmo para que o método possa convergir (próximo de $0,0001\text{ ms}$ no modelo de Bondarenko et al. [2004]). A utilização de um passo de tempo pequeno exige que um grande número de iterações seja realizado para resolver o problema, o que normalmente resulta em um tempo de solução maior. Para modelos celulares mais complexos, que descrevem com mais detalhes os processos fisiológicos, o tempo de solução dos sistemas de EDOs pode demorar ainda mais. Em nosso caso, como estamos propondo a simulação de tecidos cardíacos compostos por várias células, o método de Euler pode levar muitas horas ou até mesmo alguns dias para simular uma única batida do coração, por exemplo. Para acelerar a solução destes sistemas de equação, utilizamos neste trabalho um método numérico adaptativo no tempo desenvolvido por Campos et al. [2011]. Esse método apresenta melhor desempenho, quando comparado com o método de Euler clássico, sem inserir um erro numérico significativo. Além disso, esse método adaptativo se mostrou mais eficiente até mesmo que métodos implícitos muito utilizados para resolução de EDOs do tipo *stiff*, como o método *Backward Differentiation Formulas* (BDF). O objetivo do método é adaptar o passo de tempo, a fim de reduzi-lo em regiões mais instáveis, onde realmente ha necessidade de passos de tempo pequenos e aumentá-lo em regiões estáveis.

3.5 Combinando as estratégias de aceleração

Também faz parte da nossa proposta a combinação das técnicas descritas acima a fim de extrair todo o potencial de uma máquina paralela equipada com uma placa gráfica

e acelerar o máximo que pudermos as simulações da eletrofisiologia cardíaca.

Neste trabalho utilizamos técnicas de programação paralela em memória compartilhada para acelerar algumas operações utilizadas com a malha ALG. Além disso, implementamos em GPU os métodos numéricos para a resolução de EDOs, a fim de aumentar o desempenho das resoluções dos sistemas presentes nas simulações. Tanto os métodos com passo de tempo fixo quanto os adaptativos foram implementados e testados.

Capítulo 4

Implementações e resultados obtidos

Neste capítulo detalharemos as estratégias propostas no capítulo 3 bem como mostraremos os resultados obtidos com a implementação das mesmas.

4.1 Utilização de GPUs

Nesta seção iremos apresentar detalhadamente as implementações e resultados das estratégias de aceleração utilizando placas de processamento gráfico.

4.1.1 Implementações

Iremos descrever nesta seção as implementações dos algoritmos que utilizam GPUs com o intuito de acelerar as simulações da eletrofisiologia cardíaca. O código que implementa a solução numérica do problema monodomínio (equação (2.32)) usando o modelo de célula cardíaca LRI foi escrito em ANSI C e em paralelo usando OpenMP. As implementações GPU foram estendidas a partir do código de CPU para os diferentes ambientes de computação em GPU, a saber: OpenGL, NVIDIA CUDA para C e OpenCL. Assim, funções específicas foram desenvolvidas para cada ambiente para resolver: (i) o sistema de EDOs e (ii) o problema parabólico. Para as EDOs o método de Euler explícito foi desenvolvido, enquanto que para o problema parabólico os blocos de construção (operações com vetores, tais como SAXPY, produto escalar e multiplicação de vetores e matrizes esparsas) do algoritmo Gradiente conjugado pré-condicionando (GCP) utilizando como preconditionador o método de Jacobi foram escritos para cada código de GPU. Os detalhes de cada aplicação GPU são descritos abaixo. Vale ressal-

tar que, nesta implementação, utilizamos o método dos elementos finitos (MEF) para a discretização espacial das equações envolvidas. A figura 4.1 ilustra o algoritmo para a solução do modelo monodomínio usando a GPU, descrevendo alocações de memória, computações e transferência de dados entre CPU e GPU.

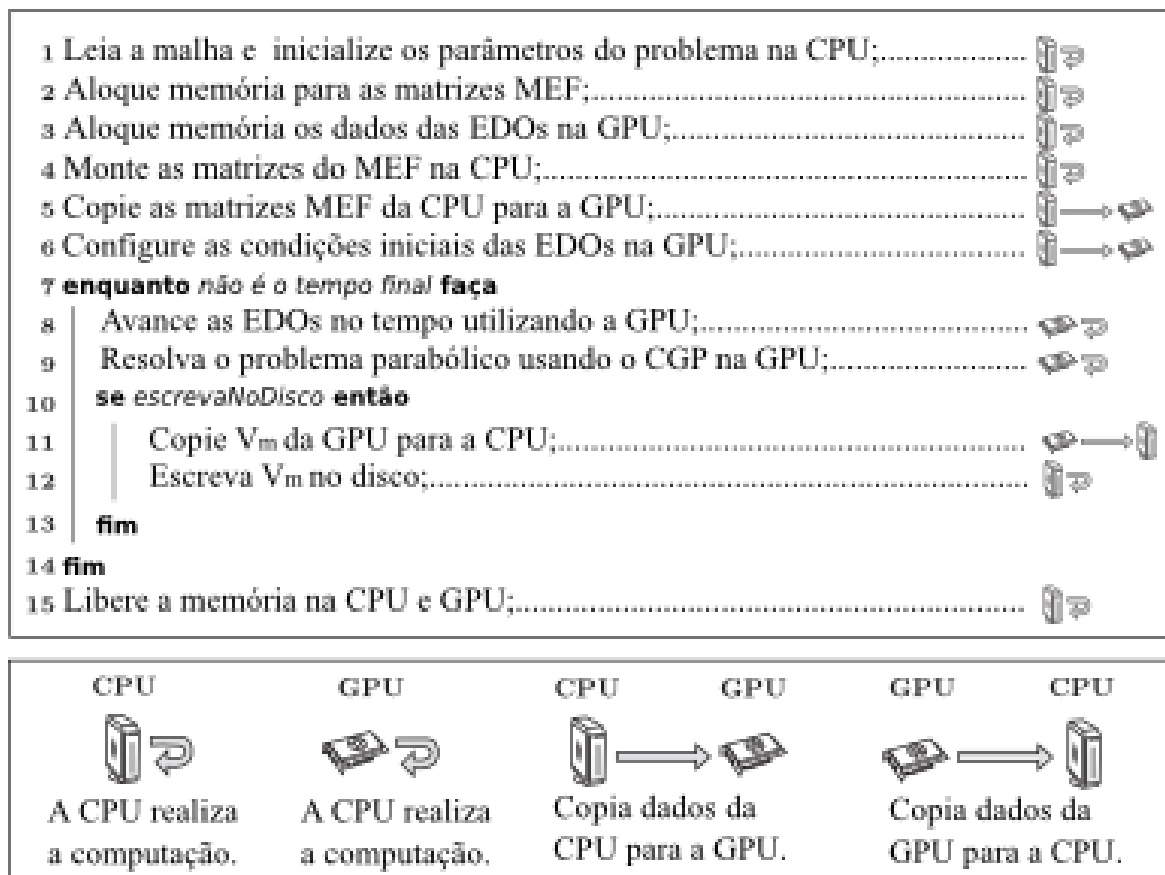


Figura 4.1. Algoritmo para a solução do modelo monodomínio.

4.1.1.1 Armazenamento das matrizes esparsas

Neste trabalho apenas malhas regulares quadradas 2D foram consideradas, resultando em matrizes esparsas com apenas 9 diagonais. A fim de se armazenar as matrizes de forma eficiente, dois formatos para matrizes esparsas foram utilizados: o formato *compressed sparse row* (CSR) e o formato *diagonal* (DIA) [Y. Saad, 1996; N. Bell & M. Garland, 2008]. O formato de CSR foi usado para montar as matrizes de elementos finitos e, em seguida, as simulações foram realizadas usando CSR ou DIA.

Sejam N e Nz o número de linhas da matriz e o número total de entradas diferentes de zero, respectivamente. Então, o formato CSR armazena uma matriz esparsa em três vetores: `vals` e `cols` que são ambos de tamanho Nz e armazenam, respectiva-

mente, as entradas não-zero e os índices de colunas dessas entradas e finalmente, um terceiro vetor chamado `PTRs` de tamanho $N + 1$ que armazena ponteiros para o início de cada linha de `vals` e `cols`.

O formato diagonal utiliza 2 matrizes para armazenar a matriz esparsa: a primeira é uma matriz retangular de $N \times Nd$ entradas, onde Nd é o número de diagonais, enquanto a segunda, denominada de *offsets*, é uma matriz uni-dimensional que armazena o deslocamento da diagonal em relação à diagonal principal.

4.1.1.2 Implementação usando OpenGL

Esta implementação utiliza os mesmos recursos de programação disponíveis para a computação gráfica, portanto, uma melhor compreensão desta abordagem requer um conhecimento profundo de computação gráfica. A ideia principal por trás da computação GPGPU usando OpenGL é mapear alguns dos conceitos de computação gráfica em conceitos de programação da CPU.

Basicamente, em computação gráfica, as texturas são usadas como dados de entrada para a renderização. Então, no nosso caso, esta máquina de renderização de gráficos é usada para executar a computação de propósito geral. Portanto, para a nossa abordagem GPGPU, as texturas foram usadas para fornecer os dados de entrada necessários para a computação, enquanto o processo de renderização realiza o cálculo que produz os resultados a serem gravados no *framebuffer*.

O *hardware* gráfico é implementado como um pipeline, e, em GPUs recentes, há algumas etapas deste pipeline que podem ser programadas. Por exemplo, o processador de vértice e de fragmento. Nós usamos o processador de fragmento em nossa implementação, porque ele se encaixa melhor para o problema e é o processador mais poderoso da GPU. O processador de fragmento executa um *shader* de fragmento, que é um programa escrito para o processador de fragmentos. O *shader* de fragmento pode ser escrito usando linguagens de *shading*, como o *OpenGL Shading Language* (GLSL) e é responsável por calcular a cor dos pixels individuais. O *shader* só pode escrever em sua própria posição, ou, em outras palavras, escritas aleatórias e dispersas não são possíveis. Texturas, no entanto, podem ser acessadas aleatoriamente, embora seja melhor manter o acesso com uma determinada localidade por causa do cache de texturas. A figura 4.2 mostra os estágios de GPUs atuais [Owens et al., 2007].

Já na figura 4.3 temos o esquema simplificado da interação entre CPU e GPU para o *loop* do método dos gradientes conjugados preconditionado (retirado de Amorim [2009]). Este é um resumo muito simplificado da programação GPU usando OpenGL. Há muito mais detalhes envolvidos nesta tarefa complexa, que não discutimos aqui.

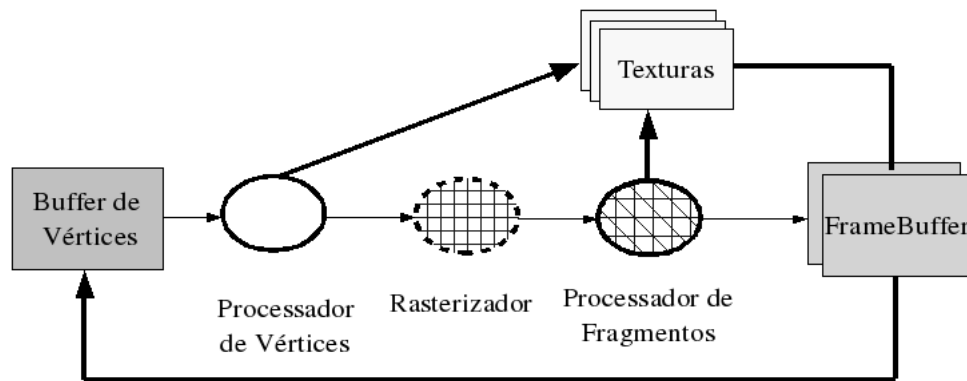


Figura 4.2. *Pipeline* Gráfica moderna. Tanto os processadores de vértices quanto o de fragmentos são programáveis.

Para uma discussão mais aprofundada sobre a aplicação OpenGL veja Amorim et al. [2009].

4.1.1.3 Implementação utilizando NVIDIA C para CUDA

O modelo de programação paralela CUDA estende a linguagem C com um conjunto minimalista de abstrações para expressar o paralelismo. Ou seja, ele permite que programador foque em questões importantes de paralelismo ao invés de lidar com conceitos complexos inerentes à computação gráfica (como é o caso quando se usa OpenGL para programação de GPU), a fim de explorar o poder computacional das GPUs para computação de propósito geral.

Um programa CUDA é organizado como um programa executando na CPU e uma ou mais funções especiais paralelas em C denominadas *kernels* que são adequadas para execução em um dispositivo de processamento paralelo, como a GPU. Um kernel executa um programa sequencial em um conjunto de *threads* paralelas. As *threads* são organizadas pelo programador em uma grade de blocos de *threads*. Todas as *threads* dentro um único bloco estão autorizadas a sincronizar umas com as outras através de barreiras e ter acesso a uma memória compartilhada por bloco de alta velocidade, que permite a comunicação entre as *threads*. *Threads* de blocos diferentes na mesma grade podem se coordenar apenas através de operações em uma memória global visível a todas as *threads*. A figura 4.4 mostra um esquema de como as *threads* são organizadas em um programa CUDA. A plataforma CUDA exige que os blocos sejam independentes, o que significa que um *kernel* deve executar corretamente, não importa a ordem em que os blocos estiverem programados para executar.

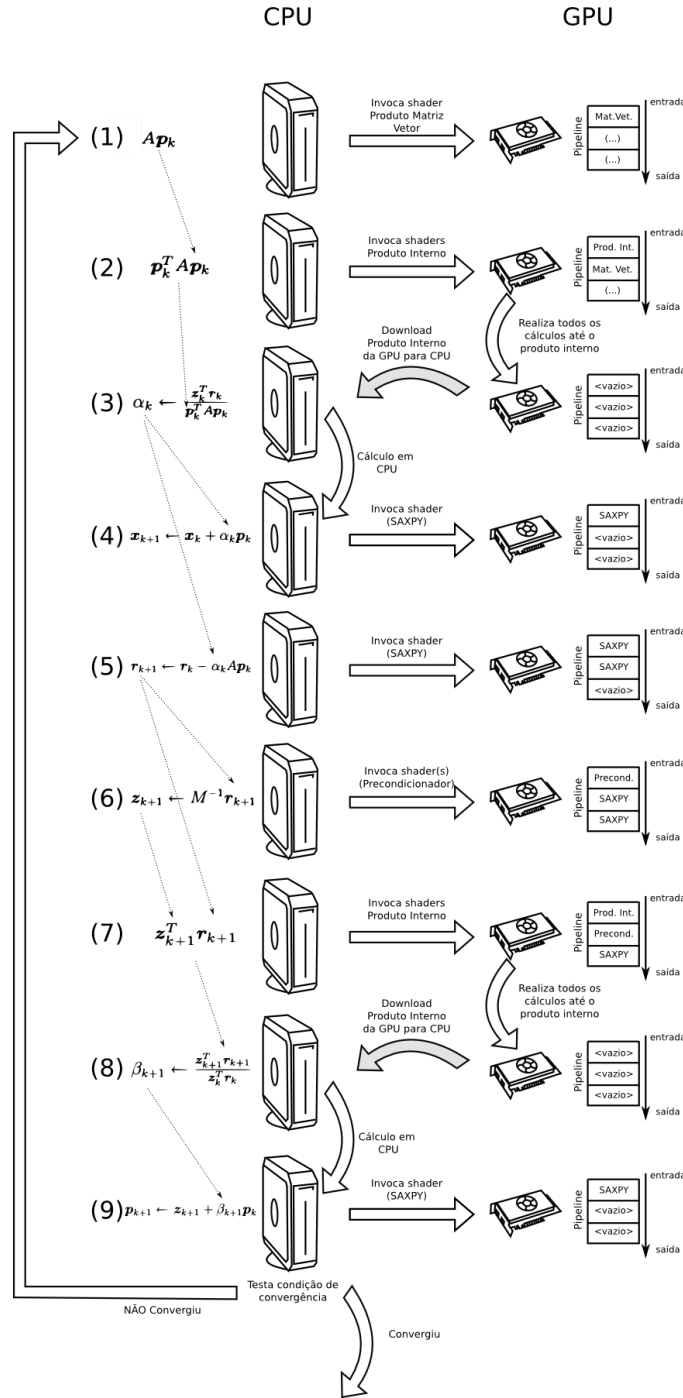


Figura 4.3. Esquema da interação entre CPU e GPU para o *loop* do GCP.

O algoritmo para a solução das equações monodomínio usando CUDA foi apresentado anteriormente em Rocha et al. [2010] e segue o esquema da figura 4.1.

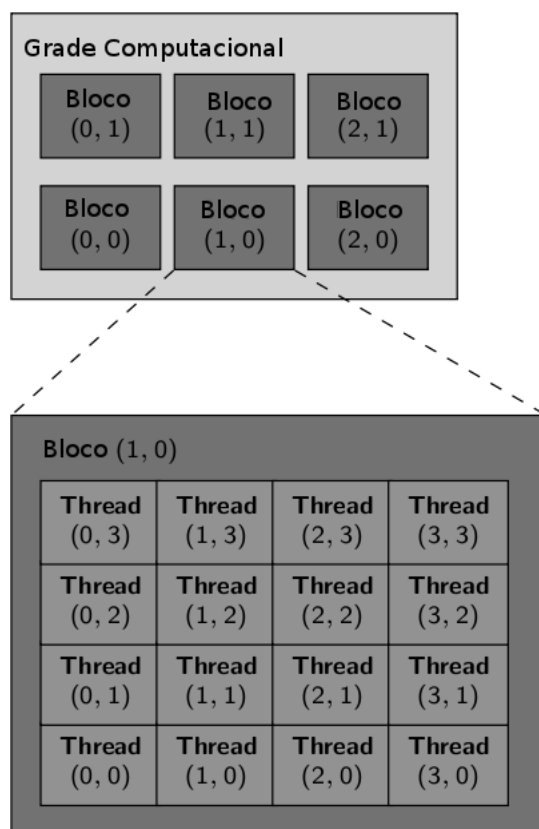


Figura 4.4. Organização das *threads* em um programa CUDA.

4.1.1.4 Implementação utilizando OpenCL

OpenCL é um padrão aberto que pode ser usado para programar CPUs, GPUs, e outros dispositivos de diferentes fornecedores. Nosso algoritmo OpenCL foi baseado no algoritmo para a solução do monodomínio utilizando CUDA apresentado anteriormente em Rocha et al. [2010]. Nos baseamos em código implementado previamente, principalmente para tornar as comparações justas. Além disso, a conversão de um kernel que usa C para CUDA para um kernel OpenCL envolve pequenas alterações.

A maior parte do código que teve que ser reescrito foi o código que executa na CPU, por exemplo, para detectar e configurar a GPU ou para copiar dados entre a CPU e o dispositivo. A tabela 4.1 mostra algumas das mudanças que tivemos que realizar no código do kernel CUDA para que ele pudesse compilar e executar sob OpenCL.

Tabela 4.1. Diretivas de *Kernel* em CUDA C e OpenCL

Mudança	<i>Kernel</i> CUDA	<i>Kernel</i> OpenCL
Definições de tipo	<code>__shared__</code> , <code>__global__</code> , etc.	<code>__local</code> , <code>__global</code> .
Indexação da <i>thread</i>	<code>threadIdx</code> , <code>blockIdx</code> , etc	<code>get_local_id()</code> , <code>get_global_id()</code> , etc
Sincronização de <i>Thread</i>	<code>__syncthreads()</code>	<code>barrier()</code>
Declaração do <i>Kernel</i>	<code>__global__ void...</code>	<code>__kernel void...</code>

4.1.2 Métodos

Os casos de teste para avaliarmos o desempenho do código consistiram em três diferentes preparações de tecido *in silico* em que as fibras foram configuradas ao longo da direção longitudinal e a discretização espacial foi definida como $\Delta x = 50 \mu m$. As simulações foram realizadas por 200 ms, após um estímulo inicial, usando um passo de tempo de $\Delta t = 0,01 ms$, o que garante a estabilidade do esquema de Euler explícito para o modelo LRI.

A tabela 4.2 apresenta as dimensões dos tecidos simulados, bem como as propriedades das malhas de elementos finitos. A tolerância relativa do método do gradiente conjugado pré-condicionado (GCP) foi configurada para 10^{-8} . Os parâmetros do modelo monodomínio foram retirados da literatura [Sundnes, 2006]: $C_m = 1 \mu F/cm^2$; $\beta = 2000 cm^{-1}$. A condutividade do tecido foi considerada homogênea com tensores condutividade dados por $\sigma_l = 3.0 mS/cm$, $\sigma_t = 1.0 mS/cm$ e $\sigma_n = 0.31525 mS/cm$.

Tabela 4.2. Detalhes da preparação *in-silico* dos tecidos e propriedades das malhas

Tecido	Nós	Elementos	Não-zero
$10 mm \times 10 mm$	200×200	39.601	357.604
$20 mm \times 20 mm$	400×400	159.201	1.435.204
$40 mm \times 40 mm$	800×800	638.401	5.750.404

4.1.3 EDOs e o problema parabólico

As variáveis de estado associadas ao sistema de EDOs de m células foram armazenadas em um vetor unidimensional de tamanho mn denominado SV , onde n é o número de equações diferenciais do modelo iônico ($n = 8$ no modelo LRI, conforme C. Luo & Yoram Rudy [1991]). Na implementação para a CPU as variáveis de estado das células são armazenadas em SV adotando uma por ordenação linha, ou seja, as primeiras 8 entradas de SV são as variáveis de estado associadas às primeiras células. Durante a fase de solução do problema um vetor contendo o potencial transmembrânico (V_m) de cada nó tem que ser passado como argumento para o método GCP. A fim de evitar movimentações extras de memória nas implementações GPU, nós reorganizamos SV usando uma ordenação por coluna, onde as primeiras m entradas dessa matriz representam os valores de V_m cada nó de discretização. A implementação GPU do passo tempo das EDOs utiliza uma *thread* para resolver um sistema de EDO, isto é, cada *thread* é associada a cada nó.

A solução do problema parabólico consiste da multiplicação de uma matriz esparsa por um vetor e a solução de um sistema de equações lineares, que também requer operações básicas de álgebra linear como o produto escalar de vetores e operações SAXPY. A implementação em GPU destes blocos de construção de álgebra linear é uma tarefa simples uma vez que cada *thread* (ou *work item* na terminologia do OpenCL) é encarregada de fazer o cálculo de uma posição da matriz, enquanto que para o produto escalar uma redução é necessária. O cálculo da multiplicação entre a matriz esparsa e o vetor $y = Ax$ usando o formato da CSR ou o formato DIA foram realizadas atribuindo uma linha da matriz para cada *thread*, que então computava uma entrada da saída do vetor y .

4.1.4 Erro numérico

Para garantir que não houve desvios nas soluções numéricas devido ao uso de precisão simples e opções de otimização do compilador, comparamos os resultados com soluções de referência obtidos na CPU usando aritmética de ponto flutuante com precisão dupla. Aqui utilizamos o EQM relativo, como mostrado na equação (4.1). Neste caso a solução de referência adotada foi a solução obtida pela execução somente em CPU ($V_{m_{cpu}}$).

$$erro = \frac{\sqrt{\sum_{i=1}^{nt} \sum_{j=1}^{nv} (V_m(i, j) - V_{m_{cpu}}(i, j))^2}}{\sqrt{\sum_{i=1}^{nt} \sum_{j=1}^{nx} V_{m_{cpu}}(i, j)^2}}, \quad (4.1)$$

4.1.5 Resultados

Nesta seção apresentamos os resultados numéricos obtidos pelas simulações monodomínio utilizando um modelo de célula ventricular em unidades de processamento gráfico através de ambientes OpenGL, CUDA NVIDIA e OpenCL. As simulações foram realizadas em um quad-core Intel Core i7 860 2.80GHz, 8GB de memória equipado com: (i) NVIDIA GeForce GTX 470 com um total de 448 núcleos CUDA, 1 GB de memória GDDR5 e 133,9 GB/s de largura de banda de memória ou (ii) AMD Radeon 6850 com 960 processadores de *Stream*, 1 GB de memória GDDR5 e 128 GB/s de largura de banda de memória.

Todo o código executado em CPU foi compilado com o compilador GNU C versão 4.4.5 usando a opção de compilação `-O3`. A versão CUDA foi desenvolvida com CUDA *Toolkit v3.2* e o código OpenCL foi desenvolvido com NVIDIA OpenCL v1.0 e

AMD APP SDK V2.4 com suporte OpenCL 1.1. Para cada teste distinguimos entre duas versões do código GPU: uma versão sem otimizações do compilador e uma versão com otimizações. A única exceção é a implementação de OpenGL, que foi tratada como código de CPU e, portanto, compilada apenas com o a opção `-O3`. As opções de compilação utilizadas para CUDA e OpenCL são diferentes. No entanto, elas foram usadas para otimizar o código tanto quanto possível com relação às operações matemáticas. Notamos aqui que o problema das EDOs tem um volume substancial de operações matemáticas que são afetados por essas opções de compilação. Para a implementação CUDA a opção `-use_fast_math` foi utilizada, enquanto as seguintes opções foram utilizados para a versão OpenCL: `-cl-fast-relaxed-math`, `-cl-mad-enable` e `-cl-no-signed-zeros`.

O desempenho do código paralelo utilizando OpenMP foi relatado anteriormente em Rocha et al. [2010]. Nesse artigo, aumentos de velocidade de 3,98 e 2,35 (comparando 4 núcleos contra um único núcleo) foram alcançados para os problemas de EDOs e parabólico, respectivamente. Nas próximas tabelas as entradas relativas ao OpenMP são os resultados de desempenho obtidos com a implementação em CPU executando com 4 núcleos de processamento.

A tabela 4.3 mostra o tempo necessário para integrar o problema das EDOs. Podemos observar que, para todos os tamanhos de problema, o código OpenGL foi melhor do que as versões OpenCL e CUDA com ou sem otimizações. Além disso, notamos que ligar as opções de otimização matemática fez com que a versão OpenCL (e CUDA) ficasse sensivelmente mais eficaz do que sem otimização. As entradas da tabela marcadas com * representam os casos que não puderam ser simulados, devido à portabilidade do código. Por exemplo, o código em NVIDIA CUDA não executa em placas gráficas AMD e nosso código OpenGL foi implementado usando recursos (texturas) que não são suportadas pelo hardware da GPU AMD.

Na tabela 4.4 descrevemos o desempenho das implementações em GPU para resolução do problema parabólico, que depende principalmente do desempenho das operações de multiplicação entre matriz esparsa e vetor, que por sua vez depende do formato de armazenamento utilizado. Fica claro que, para esta parte do problema, o ganho computacional sobre versão paralela é menor do que para a parte das EDOs, que é considerado um problema computacional embaraçosamente paralelo. Neste caso, a implementação CUDA com e sem otimizações (*-opt) superou as outras duas versões usando tanto o formato de matriz esparsa CSR quanto o DIA. É importante lembrar que o código CUDA utiliza texturas a fim de melhorar o desempenho de acesso à memória e o OpenCL não suporta esse recurso.

Os resultados mostram que, para o problema de EDOs com a maior preparação de

Tabela 4.3. Comparação entre as implementações CPU e GPU para o problema de EDOs. Os tempos de execução são dados em segundos.

GPU		GeForce GTX 470		AMD Radeon 6850	
Problema		EDO	EDO-opt	ODE	EDO-opt
200×200	OpenMP	285,65	285,65	285,65	285,65
	OpenCL	3,39	2,32	4,99	4,99
	OpenGL	2,86	2,62	*	*
	CUDA	3,57	1,63	*	*
400×400	OpenMP	1145,63	1145,63	1145,63	1145,63
	OpenCL	9,21	5,09	12,54	12,48
	OpenGL	4,11	4,09	*	*
	CUDA	11,59	4,51	*	*
800×800	OpenMP	4588,23	4588,23	4588,23	4588,23
	OpenCL	33,74	16,55	42,85	42,85
	OpenGL	10,21	10,21	*	*
	CUDA	42,59	16,03	*	*

tecido, os *speedups* obtidos (versão GPU em comparação com OpenMP com 4 núcleos) foram de 277, 286 e 446 para, OpenCL, CUDA e OpenGL, respectivamente, ao usar o hardware NVIDIA GTX 470. Um *speedup* de 107 foi obtido para o código GPU utilizando OpenCL na placa AMD Radeon 6850. *Speedups* de cerca de 4-5 foram observados para o problema parabólico usando o formato de matriz esparsa DIA tanto para OpenCL quanto OpenGL. O melhor desempenho foi obtido para a versão CUDA, onde no maior problema o código foi 8 vezes mais rápido. Um *speedup* mais modesto (2) foi obtido quando utilizamos a placa AMD Radeon 6850.

Finalmente, calculamos o erro numérico comparado com soluções de referência que foram calculadas usando o código CPU com aritmética de ponto flutuante de precisão dupla. Os erros para as implementações OpenCL e CUDA, com ou sem opções de otimização, foram menores do que 0,0050%. Os erros calculados para a implementação OpenGL foram de 0,011%. Os resultados sugerem que não houve desvios na solução nem pelo uso de precisão simples, nem pelo uso das diversas opções de otimização usadas para a compilação.

4.1.6 Discussão

Avaliamos o desempenho de diferentes algoritmos para a solução das equações monodomínio em GPU usando os ambientes paralelos OpenGL, NVIDIA CUDA e OpenCL. Três diferentes preparações de tecido *in silico* foram utilizadas para os testes de desempenho. Nós mostramos que em todos os casos, considerando a simulação completa, a GPU superou a implementação paralela de CPU usando OpenMP. Mostramos tam-

Tabela 4.4. Comparação entre as implementações CPU e GPU para o problema parabólico usando tanto o formato CSR quanto o DIA. Os tempos de execução são dados em segundos.

Versão		GeForce GTX 470			
Problema		CSR	CSR-opt	DIA	DIA-opt
200×200	OpenMP	43,02	43,02	*	*
	OpenCL	137,79	137,79	91,52	88,26
	OpenGL	216,38	205,69	135,63	135,63
	CUDA	66,03	65,47	20,64	20,64
400×400	OpenMP	254,03	254,03	*	*
	OpenCL	305,15	305,15	116,70	116,70
	OpenGL	505,45	503,84	173,42	172,61
	CUDA	225,79	225,44	47,32	47,32
800×800	OpenMP	1285,11	1285,11	*	*
	OpenCL	971,64	970,47	263,88	263,88
	OpenGL	1562,95	1562,95	312,52	312,52
	CUDA	850,21	850,21	149,16	149,16
Versão		AMD Radeon 6850			
200×200	OpenCL	272,93	272,93	246,27	235,72
400×400	OpenCL	477,79	477,12	311,63	306,45
800×800	OpenCL	1242,22	1242,22	589,24	589,24

bém o fato de que o desempenho da GPU depende da linguagem de programação para GPU adotada. Por exemplo, no problema das EDOs o código OpenGL alcançou um desempenho excelente, mesmo quando comparado às outras duas abordagens em GPU estudadas neste trabalho. No entanto, a abordagem OpenGL é difícil de programar, manter e entender, devido à complexidade em mapear os conceitos do problema nos conceitos de computação gráfica para computação GPGPU. Também é mostrado que para o problema parabólico o código CUDA foi o mais eficaz entre todas as implementações em GPU.

O código OpenCL foi um pouco mais lento do que a versão CUDA para a solução da EDPs parabólica. No entanto, para a solução do sistema de EDOs, o desempenho do OpenCL e CUDA foram muito semelhantes. Considerando o fato de que OpenCL é um padrão aberto que pode ser executado em dispositivos de aceleração diferentes (como GPUs e processadores Cell), isto sugere que OpenCL é uma alternativa promissora para que a programação de alto desempenho para computação científica seja portátil. Vale ressaltar que em todas as implementações paralelas o número de iterações do GC não apresentou variações significativas em relação ao código sequencial.

4.2 Utilização da malha adaptativa ALG

Nesta seção iremos mostrar os algoritmos relacionados a estrutura de dados da malha ALG, bem como as equações do MVF quando utilizamos esta malha e os resultados obtidos pela utilização desta estratégia de aceleração.

4.2.1 A estrutura de dados ALG

Como dito anteriormente, ALG significa *Autonomous Leaves Graph* e sua estrutura de dados é composta por dois tipos de nós: **nós células**, que são as células da malha propriamente ditas, e os **nós de transição**, que são utilizados para conectar as células com níveis de refinamento diferentes. Considerando uma implementação bidimensional, temos quatro ponteiros em cada nó célula: leste, oeste, norte e sul. Esses ponteiros podem conectar um nó célula tanto com outro nó célula ou com um nó de transição. Os nós de transição possuem apenas três ponteiros, para a conexão com nós células e/ou nós de transição. Nós células possuem variáveis adicionais que correspondem às coordenadas espaciais dos seus centros e seus respectivos estados físicos (normalmente a grandeza de interesse). Os nós também possuem informações sobre seus tipos (nó célula ou de transição) e seus níveis de refinamento.

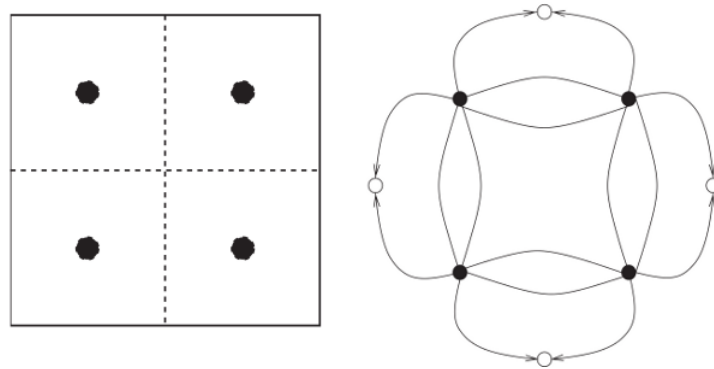


Figura 4.5. Quadrado unitário e ligações do grafo (adaptado de Burgarelli et al. [2006]).

A figura 4.5 mostra o grafo resultante após a discretização de um quadrado unitário. Este grafo possui quatro nós células com ligações orientadas ao longo das quatro direções: norte, ao longo da direção y positiva; sul, ao longo da direção y negativa; leste, ao longo da direção x positiva e oeste, ao longo de x negativo. As ligações remanescentes que não apontam para um dos quatro nós no quadrado (ou seja, as fronteiras) são então direcionadas para os quatro nós de transição, mostrados como círculos brancos.

Cada nó célula tem também dois ponteiros especiais chamados *next* e *previous*. Estes ponteiros são utilizados para ordenar os nós células no grafo. Uma vez que o grafo pode tornar-se muito irregular após sucessivos refinamentos de diferentes profundidades em diferentes regiões do domínio, pode se tornar muito difícil visitar as células de uma forma sistemática. Esta lista encadeada auxiliar facilita o acesso aos nós células, independentemente de qualquer configuração topológica que o grafo possa assumir. A manutenção desta lista e a definição da ordem de cada nó célula é feita através da curva de Hilbert modificada, que foi proposta em Burgarelli et al. [1995] e descrita detalhadamente em Burgarelli [1998].

A curva de Hilbert é uma curva de preenchimento espacial [Sagan, 1994] capaz de passar através de todos os pontos de uma malha irregular bi ou tridimensional, independentemente de quão irregular a malha pode ser. Sua implementação pode ser feita através de uma lista duplamente encadeada, utilizando os ponteiros *next* e *previous*, citados anteriormente. Qualquer célula da malha pode ser alcançada percorrendo a lista encadeada definida pela curva de Hilbert.

4.2.2 Refinamento da malha

O refinamento da malha ALG é implementado através da substituição da estrutura básica de um nó célula que irá ser refinado pela estrutura desenvolvida para o quadrado unitário, tal como apresentado acima. O nó central da célula a ser refinada é substituído por quatro nós. As quatro ligações partindo dos nós de transição são conectadas aos nós das quatro ligações originais. A figura 4.6 mostra o grafo resultante após o refinamento da célula noroeste, o número colocado ao lado de cada nó corresponde ao seu nível de refinamento.

Quando uma célula é refinada, as quatro novas células do cacho precisam ser ordenadas e inseridas na curva de Hilbert, para que possam ser acessíveis posteriormente. Essas células são ordenadas utilizando uma pequena lista, de acordo com a posição relativa da célula refinada na curva de Hilbert do grafo (antes do refinamento), que é depois inserida na lista principal. Este pré-ordenamento em uma sublista garante que as células do mesmo cacho estarão próximas umas das outras na ordenação da curva, realizando apenas uma alteração local na curva. Sendo assim, a inserção das novas células na curva principal consiste somente em inserir uma sublista já ordenada na posição da célula refinada.

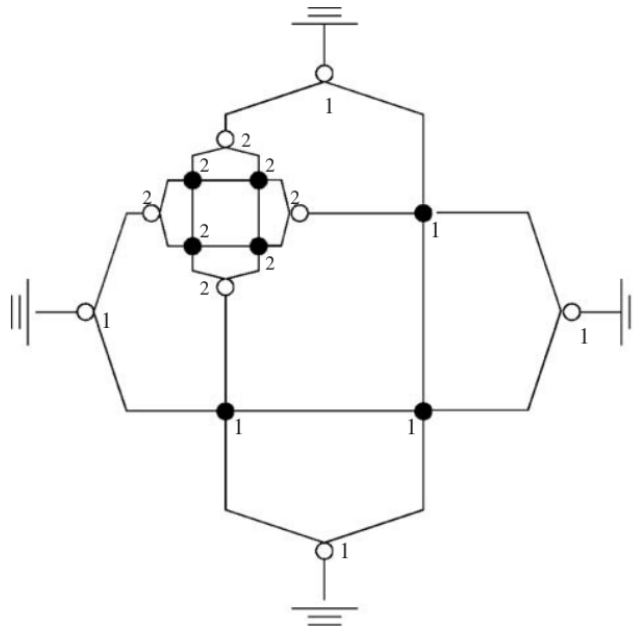


Figura 4.6. Refinamento da célula noroeste. (adaptado de Burgarelli et al. [2006]).

4.2.3 Desrefinamento da malha

Para que seja possível preservar a estrutura básica de construção do grafo, é necessário que todas as quatro células que serão desrefinadas estejam no mesmo cacho, permitindo a restauração da configuração anterior. Esta condição é necessária para assegurar que as configurações anteriores da malha possam sempre ser recuperadas executando mudanças locais mutuamente independentes. A figura 4.7 retrata o processo da seleção do cacho, desrefino do cacho selecionado e conexão dos nós em um operação de desrefinamento.

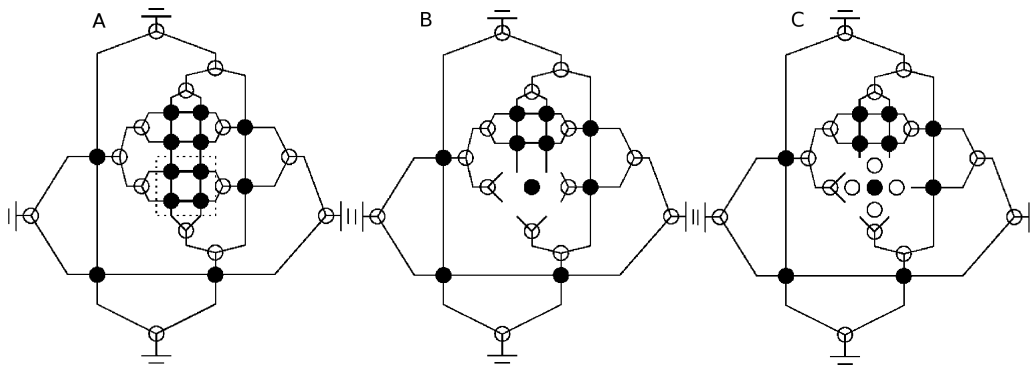


Figura 4.7. Seleção do cacho (A), desrefino do cacho selecionado (B) e conexão dos nós (C) (adaptado de Burgarelli et al. [2006]).

Após o cacho marcado na figura 4.7 A ser substituído por um único nó célula, obtemos a configuração mostrada no quadro B da figura 4.7. Os passos necessários para o desrefinamento de um cacho são:

1. Transformação do nó nordeste do cacho no nó resultante do processo de desrefinamento (denotado por v_r).
2. Preenchimento dos dados no nó resultante (por exemplo, as suas coordenadas espaciais, o nível e as constantes físicas relacionadas ao problema).
3. Conexão do nó resultante com os seus vizinhos.
4. Conexão dos nós vizinhos com o nó resultante.
5. Desalocação do espaço de memória dos nós eliminados.

A conexão do nó resultante v_r com seus nós vizinhos é feita através da criação de 4 novos nós de transição, cada um com o mesmo nível de refinamento do cacho principal. Esse procedimento pode ser visto na figura 4.7 (C).

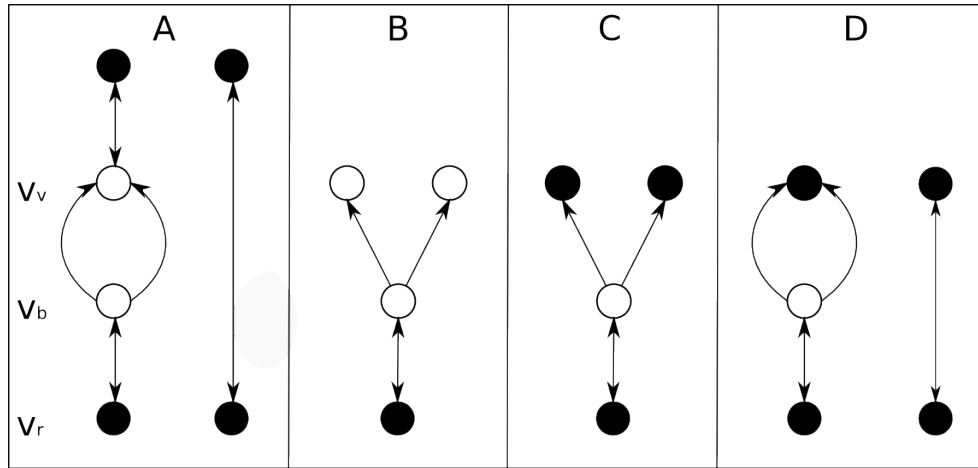


Figura 4.8. Simplificações após o refinamento (adaptado de Burgarelli et al. [2006]).

No processo de desrefinamento, um dos três ponteiros de cada um dos quatro nós de transição é apontado para o nó resultante v_r e os outros dois ponteiros são apontados para os nós vizinhos do cacho original. Para cada configuração de nós vizinhos de v_r , uma estratégia diferente para conectar esses dois ponteiros é adotada. Para as explicações a seguir, v_b denotará um dos nós de transição criados no processo de desrefinamento e v_v denotará um nó vizinho do cacho original e que se tornou adjacente ao nó v_b .

- Se v_v é um nó de transição com o mesmo nível de refinamento de v_b , após v_r ser conectado com o vizinho de v_v uma simplificação é feita para eliminar v_v e v_b (figura 4.8 A).
- Se v_v é um nó de transição e os nós v_b e v_v possuem níveis de refinamento diferentes, o nó criado v_b é mantido e será utilizado para conectar v_r com a estrutura do nó desrefinado (figura 4.8 B).
- Se v_v não é um nó de transição e os nós v_v e v_r têm diferentes níveis de refinamento, o nó de transição v_b é preservado e liga v_r à v_v (figura 4.8 C).
- Se v_v não é um nó de transição e tem o mesmo nível de refinamento de v_r , o nó de transição vb é eliminado, e o nó v_r é conectado diretamente ao nó v_v (figura 4.8 D).

Aplicando-se os passos descritos anteriormente, a malha mostrada na figura 4.7 (A), se transforma na malha da figura 4.9.

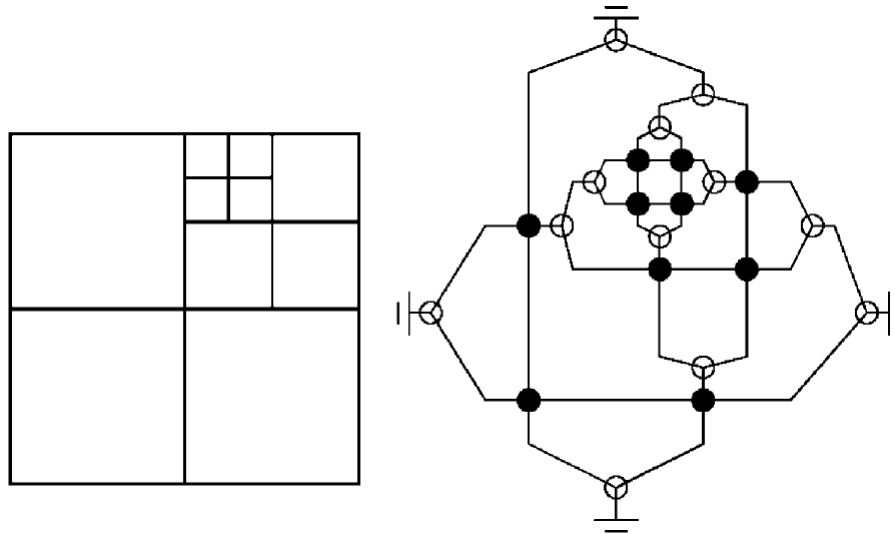


Figura 4.9. Malha após o desrefinamento de uma célula (adaptado de Burgarelli et al. [2006]).

4.2.4 ALG e o problema do monodomínio

A figura 4.10 mostra a discretização de um domínio que consiste no quadrado unitário utilizando o ALG. A enumeração dos valores $\Omega_0, \Omega_1, \Omega_2, \dots$ obedece a geometria da curva de Hilbert modificada (linha vermelha). Nesta malha exemplo, os volumes Ω_7, Ω_8 e Ω_9 estão no nível de refinamento 1; Ω_0, Ω_1 e Ω_2 estão no nível 2 e $\Omega_3, \Omega_4, \Omega_5$

e Ω_6 no nível 3. Detalhes sobre o desenvolvimento de todas as equações podem ser encontrados no apêndice A.

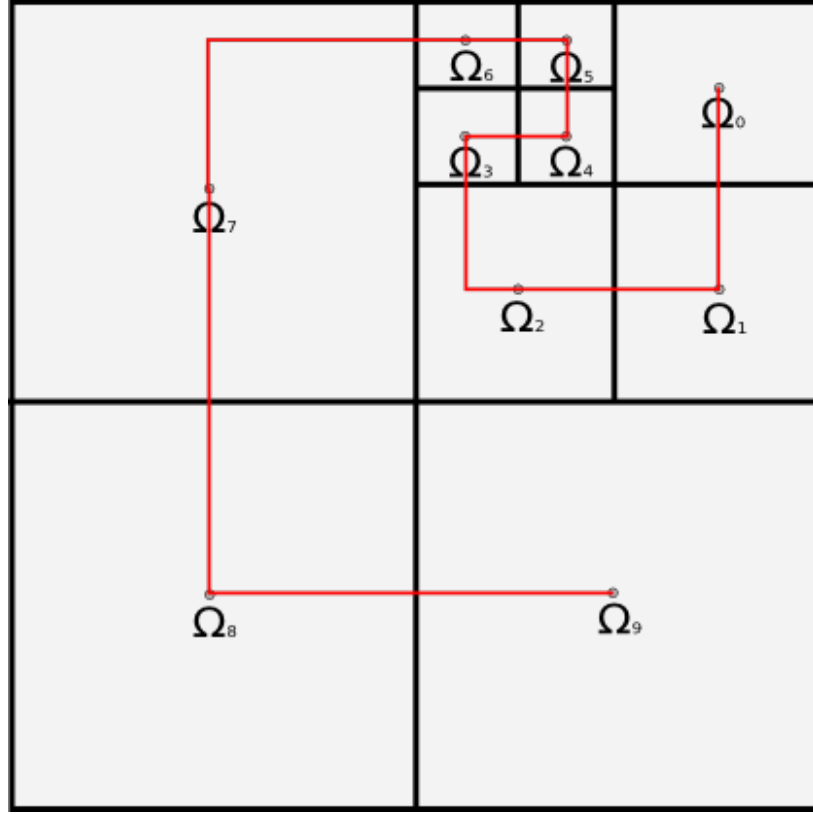


Figura 4.10. Exemplo de malha refinada usando ALG

Considerando a utilização de um tecido com as fibras orientadas horizontalmente e desenvolvendo todas as equações, temos a seguinte fórmula de avanço no tempo para os pontos interiores da malha:

$$\begin{aligned}
 & \alpha V_{i,j}^* - \\
 & \sum_{k=1}^{n_1} \sigma_{x_{d',k}} (V_{d,k} - V_{i,j}) + \\
 & \sum_{k=1}^{n_2} \sigma_{x_{e',k}} (V_{i,j} - V_{e,k}) - \\
 & \sum_{k=1}^{n_3} \sigma_{y_{c',k}} (V_{c,k} - V_{i,j}) + \\
 & \sum_{k=1}^{n_4} \sigma_{y_{b',k}} (V_{i,j} - V_{b,k}) = \\
 & V_{i,j}^n \alpha
 \end{aligned} \tag{4.2}$$

onde $\alpha = \frac{\beta C_m h_{i,j}^2}{\Delta t}$. Já nas bordas consideramos $J_{i,j} = \sigma_{i,j} = 0$ quando os pontos i,j estão fora da malha.

Usando a equação 4.2, podemos escrever os fluxos para fora de cada célula malha representada na figura 4.10, obtendo a matriz M (considerando todas as condutividades iguais a 1).

$$M = \begin{pmatrix} \alpha + 3 & -1 & 0 & 0 & -1 & -1 & 0 & 0 & 0 & 0 \\ -1 & \alpha + 3 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & -1 & \alpha + 5 & -1 & -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & 0 & -1 & \alpha + 4 & -1 & 0 & -1 & -1 & 0 & 0 \\ -1 & 0 & -1 & -1 & \alpha + 4 & -1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & -1 & \alpha + 3 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & -1 & \alpha + 3 & -1 & 0 & 0 \\ 0 & 0 & -1 & -1 & 0 & 0 & -1 & \alpha + 4 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & \alpha + 2 & -1 \\ 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & -1 & \alpha + 3 \end{pmatrix}$$

O algoritmo 1 descreve os passos utilizados para a solução numérica do modelo monodomínio utilizando a malha adaptativa ALG. Como pode ser visto, remontamos

Algoritmo 1 Passos para a solução numérica do modelo monodomínio

- 1: Para cada volume configure as condições iniciais do modelo celular;
 - 2: Configure as condições iniciais do monodomínio;
 - 3: Monte a matriz do monodomínio (sistema linear proveniente da EDP);
 - 4: **while** $t \leq t_{final}$ **do**
 - 5: Para cada volume atualize o vetor de estados do modelo celular;
 - 6: Para cada volume resolva o sistemas de EDOs referente ao modelo celular;
 - 7: Resolva o sistema linear (EDP) utilizando gradiente conjugado;
 - 8: Refine-desrefine;
 - 9: **if** refinou ou desrefinou **then**
 - 10: Remonte a matriz do monodomínio;
 - 11: **end if**
 - 12: $t = t + dt$;
 - 13: **end while**
-

a matriz do monodomínio em um determinado passo de tempo se uma operação de refinamento ou desrefinamento tiver sido realizada no mesmo passo. Neste trabalho, os critérios utilizados para refinamento e desrefinamento baseiam-se no fluxo através da interface de células vizinhas. Isto é, se o valor absoluto do fluxo é maior do que

um limiar pré-definido de refinamento ($refl = 0, 10$), o programa refina esta célula, enquanto que, se o valor absoluto do fluxo de todas as quatro células de um cacho é menor do que um limiar de desrefinamento ($drefl = 0, 20$), o programa desrefina o cacho. Para a nossa aplicação, os valores para os limiares de refinamento e desrefinamento foram encontrados empiricamente.

4.2.5 Configuração dos experimentos

Nesta seção discutimos a configuração experimental das várias simulações cardíacas realizadas.

4.2.5.1 Modelos de simulação

Para calcularmos os fatores de aceleração com a utilização de malhas adaptativas, simulamos primeiramente a atividade elétrica do coração em quatro configurações diferentes de tecido bidimensional, com malhas fixas. Como estamos interessados principalmente no impacto da discretização espacial no tempo das simulações, as configurações diferem somente no valor da discretização no espaço (referenciado como h). Os valores para h utilizados foram $25 \mu m$, $50 \mu m$, $100 \mu m$ e $200 \mu m$. Já para as malhas adaptativas, utilizamos as seguintes configurações:

1. h inicial $25 \mu m$, com h máximo $100 \mu m$ e h mínimo $25 \mu m$;
2. h inicial $25 \mu m$, com h máximo $200 \mu m$ e h mínimo $25 \mu m$;
3. h inicial $50 \mu m$, com h máximo $100 \mu m$ e h mínimo de $50 \mu m$.
4. h inicial $50 \mu m$, com h máximo $200 \mu m$ e h mínimo de $50 \mu m$.

Os domínios computacionais utilizados possuíam $1,2 cm \times 1,2 cm$. Os valores utilizados para β , C_m , σ_x e σ_y foram 0,14; 1,0; 0,0012 e 0,0006226 respectivamente. O passo de discretização temporal do modelo numérico foi ajustado para $0,01 ms$ em todos os experimentos, tanto para a evolução no tempo da EDP quanto do sistema de EDOs. As simulações foram realizadas por $40 ms$, resultando em um total de 40.000 passos de tempo, após um estímulo único ser introduzido no tecido. O modelo celular utilizado foi o modelo de C. Luo & Yoram Rudy [1991] (8 EDOs). Para a solução do sistema linear associado à resolução do modelo monodomínio utilizamos o método iterativo do Gradiente Conjugado (GC) sem preconditionador.

4.2.5.2 Plataformas computacionais

As simulações foram realizadas em máquinas equipadas com dois processadores de quatro núcleos Intel(R) Xeon(R) E5620 (2.40GHz), 12GB de memória DRAM e uma GPU NVIDIA(R) Tesla C2050 com 3GB de memória RAM e 448 núcleos de processamento.

4.2.5.3 Métricas e medidas

Utilizamos algumas métricas para comparar as diversas simulações realizadas. Os EMQ relativos das soluções obtidas pelos algoritmos que usam malhas adaptativas, em comparação com a solução de referência, foram calculados com a seguinte equação:

$$erro = \frac{\sqrt{\sum_{i=1}^{nt} \sum_{j=1}^{nv} (V_m(i, j) - V_{m_{ref}}(i, j))^2}}{\sqrt{\sum_{i=1}^{nt} \sum_{j=1}^{nx} V_{m_{ref}}(i, j)^2}}, \quad (4.3)$$

onde nt é o número total de iterações no tempo, nv é o número total de volumes utilizados, $V_{m_{ref}}$ é a solução de referência e V_m é a solução obtida pela utilização das diversas configurações utilizadas. A definição da taxa de aceleração é a tradicional, isto é, a razão entre o tempo de execução do problema de referência e o tempo de execução das configurações utilizadas.

Os erros e as taxas de aceleração das diferentes simulações foram calculados utilizando a malha com discretização mais fina ($25 \mu m$) como solução de referência. Para que pudéssemos calcular os erros entre malhas com discretizações diferentes, “refinamos” as malhas para que todas ficassem com $25 \mu m$. Esse refinamento foi feito percorrendo toda a malha e transformando os volumes com mais de $25 \mu m$ em volumes menores, repetindo o valor de V_m do volume maior em todos os novos volumes criados.

Cada tempo de execução apresentado reflete um valor médio de três execuções, que não apresentaram desvio-padrão significativo. Os códigos foram instrumentados manualmente para avaliação de desempenho. Testes comparativos realizados com e sem instrumentação indicaram que o *overhead* de instrumentação foi irrelevante. Em todas as simulações deste trabalho levamos em conta o tempo para realização de entrada e saída (E/S).

As estruturas de dados e códigos mostrados nas seções 3.3 e 4.4 foram implementadas utilizando a linguagem C++, paralelizadas com OpenMP e compiladas com o compilador gcc utilizando a opção de otimização “-O3”.

4.2.6 Resultados

Nesta seção apresentamos os resultados referentes ao tempo de execução e aceleração, bem como os erros numéricos associados à execução utilizando malhas adaptativas.

4.2.6.1 Uso de malhas grossas

A utilização de malhas com discretizações espaciais maiores é uma alternativa que pode reduzir o tempo de execução das simulações. Porém, o uso dessa estratégia pode gerar resultados com erros numéricos expressivos, quando comparados ao uso de malhas mais finas. Nesta seção iremos apresentar a aceleração obtida com o uso de malhas mais grossas, bem como os erros numéricos associados a essa estratégia.

Discretização espacial (h)	Taxa de aceleração	Erro	Iterações GC
$25 \mu m$	1	-	11497
$50 \mu m$	4,86	1,36%	6914
$100 \mu m$	23,5	5,71%	5979
$200 \mu m$	140,79	15,28%	5979

Tabela 4.5. Erros e taxas de aceleração obtidas com a utilização de malhas grosseiras

A tabela 4.5 mostra as acelerações e os erros obtidos para vários tamanhos de discretização. Podemos perceber que, como era de se esperar, o uso de valores de h maiores leva a uma redução no tempo de execução das simulações. Com a malha usando discretização espacial de $50 \mu m$ obtivemos uma simulação $4,86\times$ mais rápida que a simulação de referência. Apesar de alcançarmos taxas de aceleração satisfatórias quando aumentamos ainda mais o valor de h ($140,79\times$ com $h = 200 \mu m$), temos também um crescimento dos erros calculados, chegando a 15,28% ($h = 200 \mu m$). Vale ressaltar que o erro cresce quadraticamente com o aumento de h , o que era de se esperar já que o erro do método de volumes finitos que estamos utilizando é da ordem de $\mathcal{O}(h^2)$ e $\mathcal{O}(\Delta t)$. É importante notar que, dependendo do fenômeno simulado, erros de 15,28% ou até mesmo de 5,71% podem inviabilizar a simulação ou até mesmo invalidar os resultados. Portanto, o simples aumento da discretização espacial pode não ser uma técnica adequada para a aceleração das simulações.

4.2.6.2 Uso de malhas adaptativas

O uso das malhas adaptativas tem como objetivo a aceleração das simulações, porém sem permitir que o erro cresça proporcionalmente à taxa de aceleração. Isso é possível

pois as malhas adaptativas levam em consideração a diferença de escala dos fenômenos para apresentar soluções confiáveis e eficientes.

Apresentamos na tabela 4.6 os tempos de execução, os erros obtidos e o número total de iterações realizadas pelo GC após a utilização da malha adaptativa ALG na simulação da eletrofisiologia cardíaca. Com a utilização de uma malha adaptativa com discretização espacial inicial de $25 \mu m$ e máxima de $200 \mu m$, obtivemos uma taxa de aceleração de $21,2\times$ enquanto o erro permaneceu abaixo de 1%. Utilizando apenas malhas mais grossas, teríamos que aceitar um erro de mais de 5% para conseguirmos uma aceleração de $20\times$ como mostrado na tabela 4.5. Já nas malhas com discretização espacial inicial de $50 \mu m$ e discretizações máximas de $200 \mu m$ ou $100 \mu m$, obtivemos taxas de aceleração de $83,65x$ e $15,72x$, respectivamente. Porém, os erros obtidos com essas malhas foram maiores (2,24%, 1,83% respectivamente), mas ainda se encontram em limites aceitáveis dependendo da exigência de precisão da simulação executada. Podemos observar também que houve um aumento no número total de iterações do método GC em relação a execução utilizando malhas fixas.

Tabela 4.6. Erros e taxas de aceleração obtidas com a utilização de malhas adaptativas

Discretização espacial (h)	Taxa de aceleração	Erro	Iterações GC
Mínimo $25 \mu m$. Máximo $200 \mu m$	23,98	0,74%	16.671
Mínimo $25 \mu m$. Máximo $100 \mu m$	8,39	0,57%	12.424
Mínimo $50 \mu m$. Máximo $200 \mu m$	83,65	2,24%	8.290
Mínimo $50 \mu m$. Máximo $100 \mu m$	15,72	1,83%	10.512

Já na tabela 4.7 temos os tempos totais de execução do GC e da montagem (no caso das malhas fixas) e remontagem (malhas adaptativas) das matrizes. Podemos ver que o tempo de remontagem das matrizes é, no máximo, 5% do tempo total das resoluções do sistemas lineares. Isso mostra que o custo de remontar a matriz a cada operação de refino ou desrefino é baixo e o impacto na redução do tempo de execução do GC é significativo.

Tabela 4.7. Tempos totais de montagem de matriz e execução do GC para 3 simulações distintas

Discretização espacial (h)	Tempo GC (s)	Tempo de montagem da matriz (s)
$25 \mu m$	27.307,4	0,29
Mín. $25 \mu m$. Máx. $100 \mu m$	2.679,88	101,25
Mín. $25 \mu m$. Máx. $200 \mu m$	1.197,68	48,25

A figura 4.11 mostra o resultado da simulação utilizando a malha adaptativa ALG (mín. $25 \mu m$. máx. 200). Podemos perceber claramente que a região delimitada

pela frente de onda possui uma discretização espacial menor que o restante do tecido. O código de cores representa a distribuição da tensão transmembrânica V_m para um instante de tempo determinado.

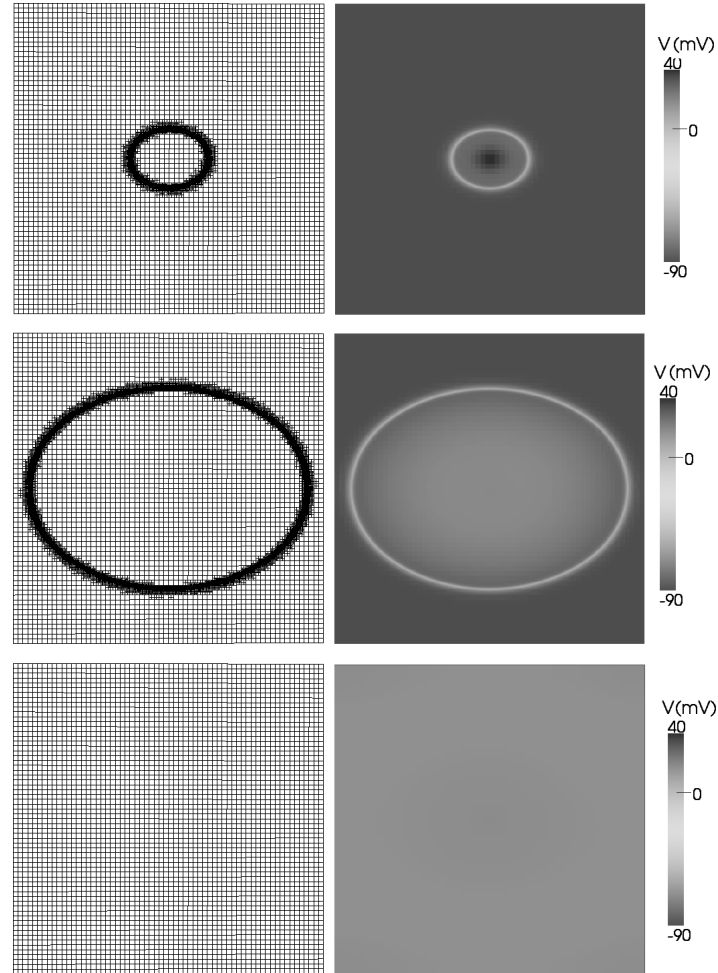


Figura 4.11. Simulação de propagação da onda elétrica (distribuição do potencial transmembrânico V_m) em tecido computacional do ventrículo em 30 ms (cima), 100 ms (meio) e 260 ms (baixo) utilizando o esquema de adaptatividade do ALG.

Na figura 4.12 apresentamos uma comparação visual entre o potencial de ação de um nó da malha computacional fixa e da malha adaptativa com o ALG (mín. 25 μm . máx. 200). Fica claro que, o erro entre as soluções de referência e a solução com a malha adaptativa usando o ALG é extremamente pequeno, como mostrado na tabela 4.6, e não afeta o comportamento da solução.

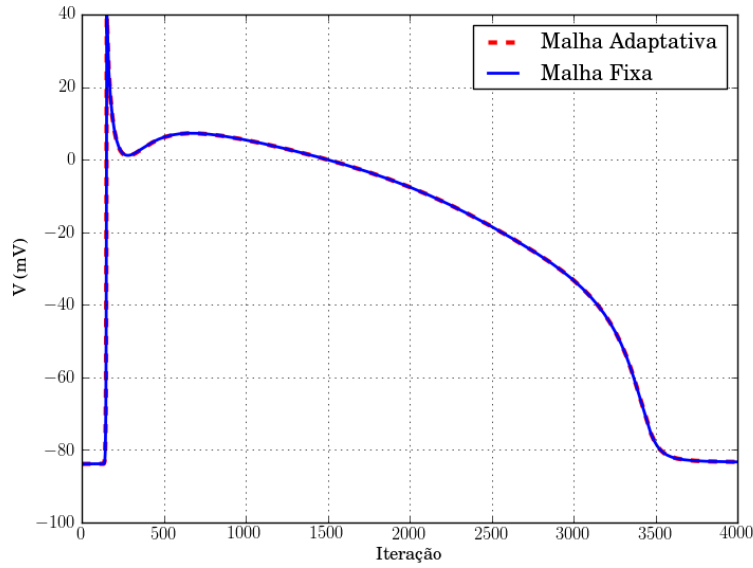


Figura 4.12. Comparação do potencial transmembrânico V ao longo do tempo em um nó da malha fixa e da malha adaptativa usando o esquema de adaptatividade do ALG.

4.2.7 Discussão

Os resultados apresentados, provenientes do uso de malhas adaptativas para aceleração das simulações da eletrofisiologia cardíaca, se mostraram extremamente promissores. A utilização da malha ALG foi eficiente tanto no que se diz respeito à taxa de aceleração (mais de $80\times$), quanto no controle da precisão numérica (erros abaixo de 2,3%).

Em relação ao método gradiente conjugado, em todos os casos houve um aumento no número total de iterações para a resolução do sistema linear a cada passo de tempo. Isso indica que a utilização de um preconditionador, como por exemplo o preconditionador de Jacobi [Shewchuk, 1994], poderia melhorar ainda mais o desempenho das simulações executadas.

4.3 Método adaptativo no tempo

Esta seção descreve o método de Euler adaptativo no tempo, utilizado na aceleração das simulações da eletrofisiologia cardíaca e também apresenta os resultados obtidos por essa abordagem.

4.3.1 Descrição e implementação

Como descrito na seção 3.4, o método de Euler explícito necessita de um passo de tempo muito pequeno para convergir. Como primeira estratégia para contornar esse problema e acelerar a resolução dos sistemas de EDOs, utilizamos intervalos de tempo diferentes para a solução dos dois problemas desacoplados, a EDP e as EDOs. Como usamos um método incondicionalmente estável para a resolução da EDP, o passo de tempo Δt_p poderia ser muito maior do que o utilizado para a solução do sistema não-linear de EDOs, $\Delta t_e = 0,0001ms$. Neste trabalho, variamos Δt_p , utilizando valores maiores que Δt_e , desde que esses valores não introduzissem erros numéricos significativos.

A segunda estratégia foi a utilização de um método de Euler adaptativo no tempo, desenvolvido por Campos et al. [2011]. O objetivo do método é adaptar o passo de tempo, tentando reduzi-lo em regiões onde há realmente necessidade de passos de tempo pequenos e aumentá-lo em outras regiões. Em nosso trabalho, optamos por calcular o passo de tempo através de uma fórmula. Essa fórmula baseia-se na comparação entre as soluções obtidas pelo método de Euler e o método de Runge-Kutta Explícito de segunda ordem (RK2) [Butcher, 2008]. O método adaptativo tem a seguinte forma, para um sistema com k equações:

$$\begin{aligned} y_{n+1}^j &= y_n^j + \Delta t_n f^j(t_n, \vec{y}_n), \quad j = 1, \dots, k \\ y_{n+1}^{rk} &= y_n^j + \frac{\Delta t_n}{2} (f^j(t_n, \vec{y}_n) + f^j(t_{n+1}, \vec{y}_{n+1})), \quad j = 1, \dots, k \\ erro_n &= \|y_{n+1}^{\vec{}} + y_{n+1}^{rk}\|_{\infty} \\ \Delta t_{n+1} &= obtem_novo_ \Delta t(erro_n, tol, \Delta t_n). \end{aligned} \quad (4.4)$$

Onde, y_{n+1}^j é a solução encontrada pelo método de Euler, y_{n+1}^{rk} é obtido utilizando o método RK2 e $obtem_novo_ \Delta t$, é a função utilizada para calcular o novo passo de tempo e $erro_n$ é a norma infinito (máximo) da diferença das soluções obtidas utilizando os dois métodos numéricos que pode ser simplificada para a seguinte forma:

$$\begin{aligned} erro_n &= \|y_{n+1}^{\vec{}} + y_{n+1}^{rk}\|_{\infty} \\ &= \|(y_n^j + \Delta t_n f^j(t_n, \vec{y}_n)) - (y_n^j + \frac{\Delta t_n}{2} (f^j(t_n, \vec{y}_n) + f^j(t_{n+1}, \vec{y}_{n+1})))\|_{\infty} \\ &= \|\frac{\Delta t_n}{2} (f^j(t_n, \vec{y}_n) - f^j(t_{n+1}, \vec{y}_{n+1}))\|_{\infty} \end{aligned} \quad (4.5)$$

Para a estimativa do novo passo de tempo é levado em consideração que $erro_n$ é da ordem de Δt_n^2 . Isto acontece pois o método de Euler é $\mathcal{O}(\Delta t^2)$ e o RK2 é $\mathcal{O}(\Delta t^3)$. Sendo assim, a diferença das soluções é $\mathcal{O}(\Delta t^2)$.

Idealmente, o novo passo de tempo encontrado deve manter o erro limitado a uma tolerância tol :

$$erro_{n+1} = \alpha \Delta t_{n+1}^2 \leq tol \quad (4.6)$$

Com $erro_{n+1} = tol$ temos:

$$\begin{aligned} \frac{erro_n}{erro_{n+1}} &= \frac{\alpha \Delta t_n^2}{\alpha \Delta t_{n+1}^2} \\ \Delta t_{n+1} &= \Delta t_n \sqrt{\frac{tol}{erro_n}} \end{aligned} \quad (4.7)$$

Se o erro obtido em uma iteração for maior do que a tolerância preestabelecida, a solução é descartada e a iteração n é refeita com o Δt_{n+1} . Caso contrário, o método obtém o novo passo de tempo e segue para a iteração $n + 1$.

O algoritmo 4.3.1 apresenta o pseudocódigo do método adaptativo utilizado neste trabalho. Observando a equação (4.4) podemos verificar a necessidade de se computar o lado direito duas vezes em cada iteração, uma para o método de Euler no instante t_n e outra para o Método RK2, no instante t_{n+1} . Se adotássemos essa estratégia dobraríamos o custo computacional deste método. Uma forma utilizada para evitar a repetição destes cálculos é reutilizar $f(t_{n+1})$ da iteração n , como $f(t_n)$ na iteração seguinte. Na primeira linha do algoritmo o lado direito é calculado e armazenado no vetor $L\vec{D}_1$, além disso o passo de tempo é inicializado com o valor do parâmetro de entrada $initial_dt$. Depois, podemos começar o método iterativo computando o método de Euler e armazenando o resultado no vetor $y_{n+1}^{\vec{}}$ (linha 3). Na linha seguinte, é computado o novo lado direito $L\vec{D}_2$ que será utilizado para calcular o erro e também na iteração seguinte para computar o método de Euler. Nas linhas 5 e 6 é calculada a tolerância tol_{n+1} . Note que se apenas o erro relativo for considerado, pode acontecer de todas as variáveis produzirem erros muito próximos de zero, o que pode ser prejudicial para a escolha da variável que possui maior instabilidade. Para amenizar este problema, esse algoritmo utiliza um esquema de tolerância variável. A cada iteração é escolhida uma tolerância variável que deve ser o maior valor entre a tolerância absoluta e o produto da tolerância relativa com a solução $y_{n+1}^{\vec{}}$, onde $relTol$ e $absTol$ são, respectivamente, as tolerâncias relativa e absoluta determinadas pelo usuário. Na linha 8 é calculado o novo passo de tempo de acordo com a fórmula apresentada na equação (4.7) e é realizada a verificação se este novo passo de tempo é maior que o passo de tempo máximo permitido, controlado pelo parâmetro de entrada max_dt . Se o novo passo calculado foi maior que max_dt o valor de entrada max_dt será utilizado como novo passo de tempo. Na linha 9 verifica-se se o erro encontrado está dentro dos limites aceitáveis,

se sim realizamos algumas trocas de variáveis para manter a consistência do método e incrementamos o tempo t . Caso o erro não seja aceitável, descartamos os cálculos pois todos os valores devem ser computados novamente.

```

1:  $t_0 = initial\_dt$ ;
2:  $L\vec{D}_1 = lado\_direito(t_0, \vec{y}_0)$ ;
3: while  $t_n \leq t_{final}$  do
4:    $\vec{y}_{n+1} = L\vec{D}_1 * \Delta t_n + \vec{y}_n$ ;
5:    $L\vec{D}_2 = lado\_direito(t_{n+1}, \vec{y}_{n+1})$ ;
6:    $tol_{n+1} = \max(relTol * \|\vec{y}_{n+1}\|, absTol)$ ;
7:    $tol_{n+1} = \max(tol_{n+1})$ ;
8:    $erro_{n+1} = \max(0.5 * \|L\vec{D}_1 - L\vec{D}_2\|)$ ;
9:    $\Delta t_{n+1} = obtem\_novo\_dt(erro_{n+1}, tol_{n+1})$ ;
10:  if  $\Delta t_{n+1} > max\_dt$  then
11:     $\Delta t_{n+1} = max\_dt$ ;
12:  end if
13:  if  $erro_n < tol_{n+1}$  then
14:     $\vec{y}_n = \vec{y}_{n+1}$ ;
15:     $L\vec{D}_1 = L\vec{D}_2$ ;
16:     $t_n = t_n + \Delta t_n$ ;
17:  else
18:    Não incrementa o tempo, para que esse passo seja feito.
19:  end if
20:   $\Delta t_n = \Delta t_{n+1}$ ;
21: end while

```

Maiores informações sobre o método adaptativo no tempo podem ser encontradas em Campos et al. [2011].

4.3.2 Configuração dos experimentos

Nesta seção discutimos a configuração experimental das várias simulações cardíacas realizadas. Vale ressaltar que as Plataformas Computacionais, Métricas e Medidas, e Metodologia foram as mesmas descritas nas seções 4.2.5.2e 4.2.5.3 respectivamente, portanto serão omitidas nesta seção.

4.3.2.1 Modelos de simulação

Para a avaliação experimental do método adaptativo no tempo, que influencia diretamente a resolução dos sistemas de EDOs, decidimos utilizar o moderno e complexo modelo de Bondarenko et al. [2004]. Este modelo descreve a atividade elétrica das células do ventrículo esquerdo dos camundongos e é baseado em um sistema linear de EDOs

com 41 variáveis diferenciais. Os parâmetros de entrada para o método adaptativo no tempo foram, para todas as simulações: $initial_dt = 0,0001\ ms$, $max_dt = 0,01\ ms$, $absTol = relTol = 10^{-6}$. Já para o método adaptativo no tempo (ALG) foram utilizadas as mesmas configurações de malhas descritas na seção 4.2.5.1.

4.3.3 Resultados

Com o objetivo de acelerar a solução dos sistemas de equações diferenciais ordinárias que fazem parte do modelo de Bondarenko et al. [2004], utilizamos neste trabalho um método numérico adaptativo no tempo desenvolvido por Campos et al. [2011]. Esse método apresenta melhor desempenho, quando comparado com o método de Euler clássico, sem inserir um erro numérico significativo.

Na tabela 4.8 podem ser vistos os tempos de execução e os erros obtidos após a utilização da malha adaptativa ALG juntamente com o método de Euler adaptativo no tempo para a resolução das EDOs. Em todas as simulações os parâmetros de entrada utilizados pelo método adaptativo no tempo foram os mesmos, como explicado anteriormente. Com a utilização de uma malha adaptativa com discretização espacial inicial de $25\ \mu m$ e máxima de $200\ \mu m$, obtivemos uma taxa de aceleração de $84,93\times$ enquanto o erro permaneceu abaixo de 2%. Já nas malhas com discretização espacial inicial de $50\ \mu m$ e discretizações máximas de $200\ \mu m$ ou $100\ \mu m$, obtivemos taxas de aceleração de $249,5x$ e $131,98x$, respectivamente. Porém, os erros obtidos com essas malhas foram maiores (2,43%, 2,03% respectivamente), mas ainda se encontram em limites aceitáveis dependendo da exigência de precisão da simulação executada. É importante ressaltar que na tabela 4.8 a taxa de aceleração é sempre em relação a uma simulação de referência utilizando uma malha fixa de $25\ \mu$ e método de Euler sem adaptatividade no tempo.

Tabela 4.8. Erros e taxas de aceleração obtidas com a utilização de malhas adaptativas e método adaptativo no tempo para a solução das EDOs.

Discretização espacial (h)	Taxa de aceleração	Erro
Mínimo $25\ \mu m$. Máximo $200\ \mu m$	84,93	1,96%
Mínimo $25\ \mu m$. Máximo $100\ \mu m$	67,15	1,63%
Mínimo $50\ \mu m$. Máximo $200\ \mu m$	249,50	2,43%
Mínimo $50\ \mu m$. Máximo $100\ \mu m$	131,98	2,03%

Já as taxas de aceleração da utilização dos métodos adaptativos no tempo e espaço em relação a utilização de somente adaptação no espaço podem ser observadas na tabela 4.9. Como pode ser visto, a utilização do método adaptativo no tempo

acelerou em média $7,52\times$ as simulações que utilizaram somente adaptatividade no espaço.

Tabela 4.9. Taxas de aceleração obtidas com a utilização de malhas adaptativas e método adaptativo no tempo para a solução das EDO em relação somente a utilização de malhas adaptativas.

Discretização espacial (h)	Taxa de aceleração
Mínimo $25\ \mu m$. Máximo $200\ \mu m$	8,17
Mínimo $25\ \mu m$. Máximo $100\ \mu m$	8,97
Mínimo $50\ \mu m$. Máximo $200\ \mu m$	5,87
Mínimo $50\ \mu m$. Máximo $100\ \mu m$	6,86

4.3.4 Discussão

Os resultados provenientes do uso de malhas adaptativas e o método de Euler adaptativo para aceleração das simulações da eletrofisiologia cardíaca, se mostraram extremamente interessantes. A utilização da malha ALG em conjunto com um método numérico adaptativo no tempo foi eficiente tanto no que se diz respeito à taxa de aceleração (mais de $80\times$), quanto no controle da precisão numérica (erros abaixo de 2%). Além disso a utilização do método adaptativo no tempo juntamente com ALG fizeram com que as simulações que utilizaram somente adaptatividade no espaço ficassem, em média, $7,52\times$ mais rápidas.

4.4 Implementações paralelas em CPU

Nesta seção iremos descrever os algoritmos e as estratégias de paralelização utilizadas para acelerar as simulações da eletrofisiologia cardíaca bem como apresentaremos os resultados obtidos por essa abordagem.

4.4.1 Descrição e implementação

Simulações em grande escala, tais como as resultantes da discretização espacial de um tecido, são computacionalmente caras. Por exemplo, quando uma discretização de $25\ \mu m$ é utilizada em um tecido com $1,28\ cm \times 1,28\ cm$ e o modelo de Bondarenko et al. [2004], que tem 41 equações diferenciais, é utilizado como modelo celular, um total de $512 \times 512 \times 41 = 10.747.904$ incógnitas devem ser calculadas a cada passo de tempo. Além disso, para simular $40\ ms$ da atividade elétrica cardíaca as mais de 10

milhões de incógnitas dos sistemas não lineares de EDOs e a EDP, com 262.144 incógnitas devem ser calculadas 40.000 vezes (com $\Delta t = 0,0001$ ms). Para lidar com esse alto custo computacional nós paralelizamos, utilizando OpenMP [Dagum & Menon, 1998] e memória compartilhada, as funções descritas nas linhas 3 e 9 (montagem da matriz do monodomínio), 6 (solução de EDOs), 7 (método do gradiente conjugado) no algoritmo 1.

Em todas as funções paralelizadas, a divisão de tarefas foi feita com base na curva de Hilbert (ver seção 4.2). Portanto, cada *thread* é responsável por calcular uma fração v do número total de volumes da simulação, onde v é dado por $\frac{V}{th}$, sendo V o total de volumes da malha em uma dado instante de tempo e th o total de *threads* envolvidas na execução.

Como dito anteriormente, para resolver os sistemas não-lineares de EDOs, tanto o método de Euler explícito quanto o método de Euler com passo adaptativo foram utilizados. Utilizando qualquer dos dois métodos, os sistemas de EDOs de cada volume finito $Vol_{i,j}$ pode ser resolvido de forma independente, fazendo com que esse problema seja embaraçosamente paralelo.

A implementação paralela do método de Euler com passo fixo é relativamente simples: cada *thread* é responsável por resolver uma fração v dos sistemas não-lineares de EDOs, onde v é calculado como descrito anteriormente. O trecho de código dessa implementação é apresentado no algoritmo 2. Nesse código, utilizamos a diretiva `parallel for` da biblioteca `openmp` para que todos os volumes (`numActiveCells`) sejam igualmente distribuídos entre as threads utilizadas e seus respectivos sistemas lineares de EDOs sejam resolvidos (`solve_ode_cpu()`). Podemos perceber também que no `for` interno utilizamos uma variável denominada `num_steps`. Essa variável é utilizada para implementar o método descrito na seção 2.7.1.1, quando utilizamos passos de tempo diferentes para a resolução da EDP e das EDOs e seu valor é a razão entre $\frac{\Delta t_e}{\Delta t_p}$.

Já para o método com passo de tempo adaptativo, um cuidado adicional deve ser tomado: como cada sistema de EDOs pode ter uma passo de tempo diferente, um desbalanceamento de carga pode aparecer quando distribuímos as tarefas já que os processadores que receberem as tarefas com passos de tempo maiores terminarão primeiro e ficarão ociosos até que os demais terminem. Para resolver esse desbalanceamento de carga, utilizamos uma técnica conhecida como sacola de tarefas: inicialmente é atribuída somente uma tarefa para cada *thread*. A medida que as *threads* terminam suas tarefas elas solicitam mais trabalho. O trecho de código dessa implementação é apresentado no algoritmo 3. Como pode ser visto, aqui também utilizamos a diretiva

Algoritmo 2 Trecho de código com a implementação paralela da resolução dos sistemas de EDOs com passo de tempo fixo

```
#pragma omp parallel for private(t,alpha,h)
for (int i = 0; i < numActiveCells; i++) {
    t = time;
    for(int j = 0; j < num_steps; j++) {
        if (stim) {
            solve_ode_cpu(dt_edo, activeCells[i]->od->sv, i_stim, t);
        }
        else {
            solve_ode_cpu(dt_edo, activeCells[i]->od->sv, 0.0, t);
        }
        t += dt_edo;
    }
    activeCells[i]->b = activeCells[i]->od->sv[0]*alpha;
}
```

`parallel for` da biblioteca `openmp` para a divisão das tarefas, porém com a utilização do parâmetro `schedule(dynamic)`. Esse parâmetro (`schedule(dynamic, C)`) faz com que as iterações sejam divididas em conjuntos de C iterações contínuas que são depois atribuídos dinamicamente a cada *thread* à medida que estes terminam de executar o conjunto anterior, ou seja, implementa exatamente a sacola de tarefas descrita anteriormente. Por omissão, C é 1.

Algoritmo 3 Trecho de código com a implementação paralela da resolução dos sistemas de EDOs com passo de tempo adaptativo

```
#pragma omp parallel for schedule(dynamic)
for (int i = 0; i < numActiveCells; i++) {
    if (stim) {
        solve_ode_cpu_adpt(time + dt_edp, activeCells[i]->od, i_stim);
    }
    else {
        solve_ode_cpu_adpt(time + dt_edp, activeCells[i]->od, 0.0);
    }
    activeCells[i]->b = activeCells[i]->od->sv[0]*alpha;
}
```

A montagem da matriz resultante da discretização da EDP parabólica também é um problema embaraçosamente paralelo. Nesta função, cada volume tem apenas de preencher a sua própria linha da matriz. Sendo assim, na nossa implementação paralela, cada *thread* é responsável por preencher uma fração das linhas.

O algoritmo 4 apresenta um trecho do código que implementa a versão paralela do método GC (algumas partes foram omitidas para simplificação). É interessante notar que esta implementação é totalmente integrada ao ALG. Neste algoritmo, *volumes* é um vetor de volumes e seus elementos são distribuídos entre as *threads* seguindo a enumeração da curva de Hilbert. Assim como nos códigos apresentados anteriormente, utilizamos a diretiva **parallel for** da biblioteca **openmp** para a divisão das tarefas, além de utilizarmos o parâmetro **reduction** que realiza toda sincronização necessária para as operações de redução. Apenas as posições com valores diferentes de zero de uma linha da matriz são armazenadas em uma lista linear em que o primeiro elemento é apontado por *firstElement*. É fácil ver que existem duas etapas embarçosamente paralelas (*EP step 1* e *EP step 2*) e duas não-embarçosamente paralelas (*non EP step 1* e *non EP step 2*). As etapas não embarçosamente paralelas são limitadas por uma redução necessária para o cálculo dos produtos escalares.

4.4.2 Configuração dos experimentos

Nesta seção discutimos a configuração experimental das várias simulações cardíacas realizadas. Vale ressaltar que as Plataformas Computacionais, Métricas e Medidas, e Metodologia foram as mesmas descritas nas seções 4.2.5.2 e 4.2.5.3, respectivamente, portanto serão omitidas nesta seção.

4.4.2.1 Modelos de simulação

Para a avaliação experimental da paralelização dos métodos descritos anteriormente refizemos os experimentos apresentados na seção 4.3.2 utilizando processadores com múltiplos núcleos. Todos os experimentos foram configurados para utilizarem os 8 núcleos presentes nos processadores (8 *threads*).

4.4.3 Resultados

A tabela 4.10 apresenta os resultados da execução paralela (8 *threads*) de uma simulação utilizando uma malha adaptativa com h mínimo de $50 \mu m$ e de máximo $200 \mu m$ e método de Euler adaptativo no tempo com uma malha quadrada com $6400 \mu m \times 6400 \mu m$. Nesta tabela o tempo total foi dividido entre tempo de solução da EDP via Gradiente Conjugado (GC), tempo para a solução das EDOs utilizando o método de Euler adaptativo no tempo (EDO), tempo para a montagem das matrizes (Matriz) e tempo total gasto pelas operações de refinamento e desrefinamento devido a utilização da malha adaptativa ALG (Ref/Deref).

Algoritmo 4 Implementação paralela do método GC utilizando ALG

```

while( numberOfIterations < maximumNumberOfIterations ) {
    pTAp = 0;
    //Non EP step 1
    #pragma omp parallel for private(element),reduction(+:pTAp)
    for (i = 0; i < numVolumes; i++) {
        volumes[i]->Ax = 0.0;
        element = volumes[i]->firstElement;
        while( element != 0 ) {
            volumes[i]->Ax += element->value * element->cell->p;
            element = element->next;
        }
        pTAp += volumes[i]->p * volumes[i]->Ax;
    }
    // Computes alpha.
    alpha = rTr/pTAp;
    // Computes new value of solution: u = u + alpha*p.
    //EP step 1
    #pragma omp parallel for
    for (i = 0; i < numVolumes; i++) {
        volumes[i]->v += alpha * volumes[i]->p;
    }
    r1Tr1 = 0.0;
    //Non EP step 2
    #pragma omp parallel for reduction (+ : r1Tr1)
    for (i = 0; i < numVolumes; i++) {
        volumes[i]->r -= alpha * volumes[i]->Ax;
        r1Tr1 += volumes[i]->r * volumes[i]->r;
    }
    //Computes beta.
    beta = r1Tr1/rTr;
    pError = r1Tr1;
    numberOfIterations++;
    if( pError <= precision ) {
        break;
    }
    //Computes vector p1 = r1 + beta*p
    //and uses it to upgrade p.
    //EP step 2
    #pragma omp parallel for
    for (i = 0; i < numVolumes; i++) {
        volumes[i]->p1 = volumes[i]->r + beta * volumes[i]->p;
        volumes[i]->p = volumes[i]->p1;
    }
    rTr = r1Tr1;
}

```

Tabela 4.10. Resultados da execução paralela (8 *threads*) de uma simulação utilizando uma malha adaptativa com h mínimo de $50 \mu m$ e de máximo $200 \mu m$ e método de Euler adaptativo no tempo.

	Tempo (μs)	% da execução	<i>Speedup</i>	Desvio padrão - <i>Speedup</i>
Total	88977882	100,00	7,16	0.25
GC	1343870	1,51	4,47	0,16
EDO	78503100	88,23	7,95	0,06
Matriz	887945	1,00	1,92	0,18
Ref/Deref	6121921	6,88	-	-

Como pode ser visto, obtivemos um *speedup* praticamente linear ($7,16\times$ utilizando 8 *threads*) para o tempo total de simulação em relação a versão sequencial da mesma simulação. Podemos observar também que, nesta simulação, a resolução das EDOs consome mais de 80% do tempo total de simulação. Vale ressaltar que, mesmo existindo um grande desbalanceamento inerente ao método de Euler adaptativo no tempo, conseguimos obter um *speedup* de $7,95\times$ com a nossa implementação paralela utilizando sacola de tarefas. É importante mencionar também que as operações de refinamento e desrefinamento contabilizaram somente por 6,88% do tempo total de execução.

Para testarmos a escalabilidade de nossa implementação, realizamos uma simulação utilizando uma malha computacional maior, com $102.400\mu m \times 102.400\mu m$ e h mínimo de $50\mu m$ e de máximo $200\mu m$. Como pode ser visto, o *speedup* total continuou praticamente linear ($7,77\times$). O aumento mais significativo, tanto em relação ao *speedup* quanto o percentual do tempo total de execução aconteceu no método do gradiente conjugado.

Tabela 4.11. Resultados da execução paralela (8 *threads*) de uma simulação utilizando uma malha adaptativa com h mínimo de $50\mu m$ e de máximo $200\mu m$, método de Euler adaptativo no tempo e malha com $102.400\mu m$.

	Tempo (μs)	% da execução	<i>Speedup</i>	Desvio padrão - <i>Speedup</i>
Total	1.267.934.962	100,00	7,77	0,23
CG	245.628.018	19,37	12,94	0,39
EDO	829.624.332	65,43	7,70	0,16
Matriz	35.273.691	2,78	4,21	0,085
Ref/Deref	98.619.711	7,78	-	-

Levando em consideração que, como apresentado na seção 4.3.3, com a execução sequencial utilizando a malha adaptativa juntamente com o método adaptativo no tempo obtivemos uma aceleração máxima de $249,5\times$ em relação de referência (malha de $25\mu m$), a aceleração total obtida após a paralelização foi de $1.786,42\times$. Vale ressaltar que a variação do erro numérico após a paralelização foi irrelevante.

4.4.4 Discussão

A paralelização do método do gradiente conjugado, método de Euler adaptativo e a montagem das matrizes, visando a aceleração das simulações da eletrofisiologia cardíaca, se mostrou extremamente eficiente. Os *speedups* obtidos foram praticamente lineares ($7,16\times$ com 8 *threads*) e a taxa de aceleração total foi de $1.786,42\times$, sendo $7,16\times$ obtida com a paralelização multiplicado por $249,5\times$ obtida com a utilização da

malha ALG. Vale ressaltar que esse taxa de aceleração foi alcançada utilizando somente uma estação de trabalho com um processador de 8 núcleos sem a necessidade da utilização de um *cluster* de computadores. Devido à problemas de escalabilidade, *speedups* dessa magnitude são dificilmente alcançados mesmo com a utilização de *cluster* com mais de 1.024 nós se empregando, por exemplo, decomposição de domínio e algoritmos em memória distribuída, como pode ser visto em Plank et al. [2009], onde os autores alcançaram um *speedup* de aproximadamente $76\times$ utilizando um cluster com 128 nós.

4.5 Implementação paralela utilizando GPU

Descrevemos nesta seção a paralelização em GPU dos algoritmos descritos anteriormente visando a aceleração das simulações da eletrofisiologia cardíaca bem como apresentamos os resultados obtidos por essa abordagem.

4.5.1 Descrição e implementação

Em nossa implementação utilizando GPU decidimos manter a abordagem descrita na seção 4.4 para a solução do sistema linear associado com a discretização da EDP do modelo monodomínio. Ou seja, resolvemos a EDP discretizada utilizando o método GC paralelo em CPU. No entanto, a solução dos sistemas de EDOs foi acelerada com a utilização de GPUs. Esta é uma estratégia diferente das que usamos antes, como descrito na seção 4.1, quando a solução todas as equações do monodomínio (EDP parabólica e sistema de EDOs) foi completamente implementada em GPU.

A motivação para a escolha de uma estratégia diferente é baseada em várias razões. Conforme apresentado na seção 4.1, a resolução das equações do modelo monodomínio pode ser acelerada em até 35 vezes utilizando uma única GPU quando comparada a uma implementação paralela com OpenMP executada em um computador com quatro núcleos. No entanto, o *speedup* total obtido por essa implementação em GPU é uma combinação de um *speedup* próximo de $10\times$ para a solução da EDP com um *speedups* próximo de $450\times$ para a solução dos sistemas não lineares de EDOs. Hoje em dia, com a evolução das arquiteturas com múltiplos núcleos, é possível encontrar facilmente no mercado um único computador equipado com 64 núcleos de processamento. Portanto, acreditamos que a resolução da EDP utilizando implementações paralelas baseadas em OpenMP executando sobre essas novas máquinas com vários núcleos podem superar uma implementação utilizando GPU. Por outro lado, para a solução dos sistemas não-lineares de EDOs a implementação paralela em GPU ainda supera facilmente estes novos computadores com vários núcleos. Sendo assim, optamos por implementar

em GPU somente a resolução paralela dos milhões de sistemas não-lineares de EDOs. Além disso, a natureza não regular do ALG faz com que a GPU não seja a arquitetura ideal para a sua paralelização.

Neste trabalho, a resolução dos sistemas não-lineares de EDOs segue uma abordagem semelhante à apresentada na seção 4.1.3. Para esta implementação também utilizamos o modelo de programação paralela CUDA e os modelos celulares são resolvidos da mesma maneira como descrito anteriormente (seção 4.1.3). No entanto, devido a natureza adaptativa da malha ALG, fizemos algumas modificações no algoritmo 1 para evitar transferências de dados (vetor **SV**) entre as memórias da GPU e CPU após os sucessivos refinamentos e desrefinamentos. Isso foi necessário pois é bem conhecido que transferências de memória representam um dos principais gargalos de desempenho em se tratando de programação em GPUs [Ryoo et al., 2008].

O algoritmo 5 descreve os passos utilizados para a solução numérica do modelo monodomínio utilizando computação paralela tanto em CPU quanto em GPU. Primeiramente, precisamos inicializar o dispositivo gráfico que será responsável pela execução de partes específicas do programa. Em seguida, alocamos na GPU o vetor de estados do modelo celular (**SV**). Esse vetor é alocado com tamanho grande o suficiente para armazenar os **SVs** de todas as células quando estivermos usando a menor discretização permitida, que é pré-configurada pelo usuário através de parâmetros de entrada do programa. Modificamos também a estrutura dos nós células do ALG para que estes armazenassem o índice no vetor **SVs** (alocado na GPU) no qual seu vetor **SV** local está armazenado. Além disso, utilizamos uma estrutura de dados que contém todos os índices do vetor **SVs** que não estão sendo utilizados em um determinado instante de tempo (mapa de livres). Inicialmente, o mapa de livres se encontra vazio já que todas as posições do vetor de **SVs** na GPU estão ocupadas pois sempre iniciamos a execução com a menor discretização permitida. Também modificamos os métodos de refinamento e desrefinamento para que estes executassem as seguintes tarefas:

1. Sempre que uma célula for desrefinada, os índices do vetor **SV** armazenados nas três células que estão sendo removidas devem ser adicionados ao mapa de livres.
2. Sempre que uma célula for refinada, três índices que representem posições livres no vetor **SV** devem ser retiradas do mapa de livres e armazenadas nas células em questão para que estas possam utilizar as posições livres no vetor de **SVs** na GPU para guardar seus respectivos vetores de estado.

Depois dos passos de refinamento e desrefinamento terem sido executados, percorremos a malha para construir um vetor denominado `cellsToSolve`. Esse vetor é

Algoritmo 5 Passos para a solução numérica do modelo monodomínio

```

1: Para cada volume ajuste as condições iniciais do modelo celular em paralelo (GPU);
2: Configure as condições iniciais do monodomínio em paralelo (CPU);
3: Monte a matriz do monodomínio em paralelo (CPU);
4: while  $t \leq t_{final}$  do
5:   Para cada volume atualize o vetor de estados do modelo celular (GPU);
6:   Para cada volume resolva o sistemas de EDOs referente ao modelo celular (GPU);
7:   Resolva o sistema linear (EDP) utilizando gradiente conjugado paralelo (CPU);
8:   Refine-desrefine (CPU);
9:   if refinou ou desrefinou then
10:    Atualize as posições utilizadas no vetor de estados (GPU);
11:    Remonte a matriz do monodomínio em paralelo (CPU);
12:   end if
13:    $t = t + dt$ 
14: end while

```

então copiado para a memória da GPU e armazena quais os índices do vetor **SVs** que devem ser utilizados para a resolução das EDOs. Além disso, também precisamos informar a GPU quais células foram refinadas em um determinado passo de tempo para que o vetor estado dessas células sejam atualizados corretamente.

A figura 4.13 mostra atualização do vetor **cellsToSolve** e do mapa de livres após o desrefinamento do cacho nordeste de uma malha ALG. O painel A mostra a malha inicial, com 16 elementos. Inicialmente, as EDOs de todas as células armazenadas na GPU devem ser resolvidas, por isso o vetor **cellsToSolve** contém o índice de todas as células (dado inicialmente pela curva de Hilbert) e o mapa de livres se encontra vazio. Após desrefinarmos o cacho nordeste, 3 posições do vetor de SVs da GPU ficam desocupadas e poderão ser utilizadas posteriormente. Sendo assim, o mapa de livres contém o índice das 3 células que foram removidas após o desrefinamento e agora o vetor **cellsToSolve** armazena somente os índices dos SVs que devem ser utilizados para a resolução das EDOs, de acordo com a configuração atual da malha. É importante notar que no painel B da figura 4.13, os números mais acima (em preto) se referem à numeração da célula dada de acordo com a curva de Hilbert e os números abaixo (em vermelho) são os índices no vetor SVs da GPU onde o vetor de estados da célula está armazenado.

É importante ressaltar que tanto o método de Euler com passo fixo quanto com passo adaptativo foram implementados para a resolução dos sistemas de EDOs (linha 6 do algoritmo 5).

4.5.2 Resultados

Na tabela 4.12 temos os resultados da execução paralela (8 *threads* + GPU) de uma simulação utilizando uma malha adaptativa com h mínimo de $50 \mu m$ e de máximo $200 \mu m$ e método de Euler adaptativo no tempo com uma malha quadrada num domínio

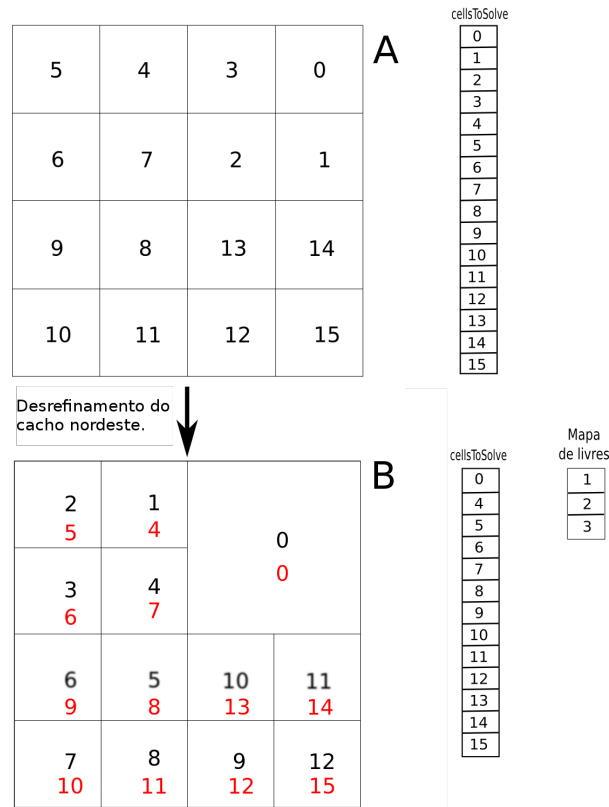


Figura 4.13. Atualização do vetor `cellsToSolve` e do mapa de livres após o desrefinamento do cache nordeste de uma malha ALG.

de $6.400\mu m \times 6.400\mu m$. Assim como feito anteriormente, dividimos o tempo total entre tempo de solução da EDP via Gradiente Conjugado (GC), tempo para a solução das EDOs utilizando o método de Euler adaptativo no tempo (EDO), tempo para a montagem das matrizes (Matriz) e tempo total gasto pelas operações de refinamento e desrefinamento devido a utilização da malha adaptativa ALG (Ref/Deref).

Nesta simulação, tanto método do gradiente conjugado quanto a montagem das matrizes foram realizadas de forma paralela, em CPU, utilizando 8 núcleos. Já a resolução do método de Euler foi realizada utilizando uma GPU. Como pode ser visto, para esta configuração de simulação, a paralelização em GPU do método de Euler adaptativo no tempo alcançou um *speedup* marginal, de apenas $1,77\times$, muito pior que a paralelização somente em CPU, que alcançou *speedups* próximos de $8\times$. Isso acontece devido ao grande desbalanceamento de carga inerente ao método adaptativo no tempo, como explicado anteriormente. É bem conhecido que o desbalanceamento de carga ainda não é tratado de forma satisfatória pelos paradigmas de programação de GPU atuais, por exemplo, o escalonador CUDA não consegue gerenciar bem cargas de trabalho desbalanceadas.

Tabela 4.12. Resultados da execução paralela (8 *threads* + GPU) de uma simulação utilizando uma malha adaptativa com h mínimo de $50 \mu m$ e de máximo $200 \mu m$ e método de Euler adaptativo no tempo.

	Tempo (μs)	% da execução	<i>Speedup</i>	Desvio padrão - <i>Speedup</i>
Total	359.515.368	100,00	1,77	0,23
GC	4.336.319	1,21	1,39	0,1
EDO	338.186.529	94,07	1,84	0,03
Matriz	628.843	0,17	2,71	0,11
Ref/Deref	8.308.344	2,31	-	-

Já tabela 4.13 temos os resultados da execução paralela (8 *threads* + GPU) de uma simulação utilizando uma malha adaptativa com h mínimo de $50 \mu m$ e de máximo $200 \mu m$ e método de Euler adaptativo no tempo com uma malha quadrada com domínio de $102.400 \mu m \times 102.400 \mu m$. Nesta tabela o tempo também foi dividido como descrito anteriormente. Nesta simulação o tecido computacional é consideravelmente maior que o utilizando na simulação anterior. Sendo assim, temos muito mais sistemas de EDOs para serem resolvidos, fazendo com que o problema de desbalanceamento de carga seja menos proeminente e permitindo a obtenção de um *speedup* de $11,93\times$ na resolução das EDOs. Nesse caso, o *speedup* obtido com a utilização de GPUs, em relação a mesma simulação executada sequencialmente, foi aproximadamente $10,29\times$. Realizando uma

Tabela 4.13. Resultados da execução paralela (8 *threads* + GPU) de uma simulação utilizando uma malha adaptativa com h mínimo de $50 \mu m$ e de máximo $200 \mu m$, método de Euler adaptativo no tempo e e malha num domínio de $102.400 \mu m \times 102.400 \mu m$.

	Tempo (μs)	% da execução	<i>Speedup</i>	Desvio padrão - <i>Speedup</i>
Total	957.533.932	100,00	10,29	0,18
GC	242.951.983	25,37	13,08	0,46
EDO	534.992.610	55,87	11,93	0,16
Matriz	33.516.526	3,50	4,43	0,22
Ref/Deref	85.710.508	8,95	-	-

aproximação baseada nos resultados obtidos anteriormente, nosso método utilizando GPU poderia alcançar mais de $2,000\times$ de taxa de aceleração em relação a simulação de referência ($25 \mu m$).

4.5.3 Discussão

A combinação das técnicas desenvolvidas neste trabalho se mostrou extremamente promissora e eficiente. Alcançamos um *speedup* de $10,29\times$, em relação a aplicação se-

quencial e os resultados mostraram que poderíamos alcançar mais de $2000\times$ de taxa de aceleração em relação a simulação de referência ($25\mu m$). Podemos observar também um *speedup* super-linear obtido no método GC. De acordo com nossas análises isso ocorreu devido ao efeito de cache, pois trata-se de uma malha com um número considerável de volumes e mais elementos podem ser armazenados na cache pois neste caso estamos utilizando a cache combinada de 8 processadores e não somente um. É muito importante ressaltar novamente que esse *speedup* foi alcançado utilizando somente uma estação de trabalho com um processador de 8 núcleos e uma GPU, sem a necessidade da utilização de um *cluster* de computadores.

A tabela 4.14 mostra as taxas de aceleração máximas obtidas pelas técnicas desenvolvidas neste trabalho. Comparando com trabalhos recentes, as taxas de aceleração obtidas com as técnicas apresentadas neste trabalho, utilizando apenas uma máquina com 8 núcleos e uma GPU, podem ser consideradas extremamente altas. Se analisarmos, por exemplo, o trabalho proposto Southern et al. [2012] onde os autores paralelizaram um algoritmo de malha adaptativa desenvolvido em Pain et al. [2001] podemos observar que taxa de aceleração total obtida, utilizando um cluster com 64 nós foi de $1.024\times$ enquanto as taxas máximas de aceleração mostradas nesta tese foram de $1.786,42\times$ utilizando uma máquina com 8 processadores e poderia alcançar mais de $2000\times$ utilizando 8 processadores + GPU. Além disso, no trabalho mencionado, o algoritmo paralelo apresenta uma eficiência paralela reduzida com mais de 64 nós.

Tabela 4.14. Taxas de aceleração das técnicas desenvolvidas

Método	Aceleração
ALG	83,65
ALG + 8 cores	632,394
ALG + Euler adaptativo	249,5
ALG + 8 cores + Euler adaptativo	1786,42
ALG + 8 cores + GPU + Euler adaptativo (estimativa)	> 2500

Levando em consideração somente a utilização da malha adaptativa ALG, ainda assim alcançamos taxas de aceleração consideráveis. As taxas de aceleração máximas obtidas com a utilização somente de malhas adaptativas descritos em Southern et al. [2010] e Southern et al. [2012] são de $11,2\times$ e $13\times$ respectivamente. Já neste trabalho, atingimos uma taxa de aceleração máxima de $83,65\times$ utilizando a malha adaptativa ALG. De acordo com nossas análises, os resultados superiores obtidos utilizando as estratégias apresentadas nesta tese provêm principalmente da eficiência das operações relacionadas a adaptatividade da malha ALG. Como pode ser visto em Southern et al. [2012], o *overhead* causado pela adaptatividade foi em média 75%, muito maior do que

o *overhead* causado pela malha ALG que foi, no pior caso, de apenas 12,45%.

Comparando com trabalhos menos recentes, também obtivemos melhores taxas de aceleração. Em Cherry et al. [2000] e Trangenstein & Kim [2004] por exemplo, a maior taxa de aceleração apresentada foram de $5\times$ e $1,72\times$ respectivamente, muito inferior as acelerações apresentadas neste trabalho.

Capítulo 5

Conclusões e trabalhos futuros

Neste capítulo apresentamos as conclusões deste trabalho bem como descrevemos seus possíveis trabalhos futuros.

5.1 Conclusões

Neste trabalho, propusemos, implementamos e combinamos estratégias de computação paralela, malhas adaptativas no espaço e métodos adaptativos no tempo para aumentar o desempenho das simulações da eletrofisiologia cardíaca.

Nossa primeira abordagem para acelerar as simulações é o uso de GPUs na resolução dos modelos matemáticos. Três diferentes implementações GPU foram comparadas, OpenGL, NVIDIA CUDA e OpenCL, a uma implementação paralela em CPU utilizando OpenMP. A abordagem utilizando OpenGL mostrou ser a mais rápida com um *speedup* de 446 (em comparação com a implementação com vários núcleos) para a solução do sistema não-linear de EDOs associada à solução do modelo cardíaco, enquanto a implementação em CUDA foi a mais rápida para a solução numérica da EDP parabólica atingindo um *speedup* de 8. A versão OpenCL foi um pouco mais lenta para a solução da equação parabólica e tão rápido quanto CUDA para a solução do sistema de EDOs, mostrando, portanto, ser uma estratégia de programação eficiente e portátil para resolver problemas científicos.

A segunda proposta foi a utilização da malha adaptativas ALG para aceleração das simulações. Os resultados obtidos indicaram que a utilização do ALG é capaz de acelerar uma simulação em mais de $83\times$, quando comparado ao uso de malhas fixas. Além disso, a precisão numérica também foi mantida sob controle (erros relativos abaixo de 3%).

Além disso, combinamos técnicas de malhas adaptativas, métodos numéricos adaptativos no tempo e computação paralela (CPU e GPU) para tentarmos acelerar ainda mais as simulações. Os resultados obtidos foram extremamente promissores, alcançando *speedups* de mais $1.700\times$ utilizando apenas uma máquina equipada com 8 processadores e uma placa aceleradora. Além disso, simulações realizadas com tecidos maiores mostraram evidências de que poderíamos alcançar uma aceleração superior a $2.500\times$. Vale ressaltar também que as taxas de aceleração apresentadas neste trabalho são expressivamente maiores as taxas obtidas por técnicas similares encontradas em publicações recentes. Isso mostra que a nossa hipótese, onde assumimos que assume que é possível realizar simulações cardíacas mais eficientes tanto em tempo de execução quanto em utilização de memória principal e secundária pela adoção de novas estratégias de solução foi comprovada com os resultados obtidos.

Todos os métodos descritos neste trabalho podem ser aplicados de maneira simples em modelos de simulação 3D. Como a complexidade de todos os métodos descritos é em função do número de nós da malha, não esperamos variações significativas nos resultados e fatores de aceleração. Na figura 5.1, podemos ver um passo de tempo da simulação de propagação da onda elétrica (distribuição do potencial transmembrânico V_m) em tecido computacional tridimensional do coração de um coelho utilizando os métodos descritos neste trabalho adaptados para simulações 3D.

É importante ressaltar também que, as técnicas desenvolvidas e avaliadas nesta tese podem ser aplicadas (realizando-se as devidas adaptações) na resolução de uma série de outros fenômenos que utilizem equações de reação-difusão na sua modelagem.

5.2 Trabalhos em andamento e futuros

Apesar de apresentarmos uma melhoria significativa no tempo de execução das simulações da eletrofisiologia cardíaca, ainda nos encontramos bem distantes de simulações em tempo real. Diversas otimizações, melhoramentos e novas estratégias podem ser utilizadas bem como as técnicas apresentadas podem ser combinadas, para que possamos atingir nossos objetivos. Além disso, este trabalho torna viável a materialização de algumas ideias que, mesmo não sendo essenciais, são atraentes tanto do ponto de vista científico como de um ponto de vista prático. Sem fazer distinção entre as diversas classes de melhoramentos, prosseguiremos detalhando-as nas subseções abaixo.

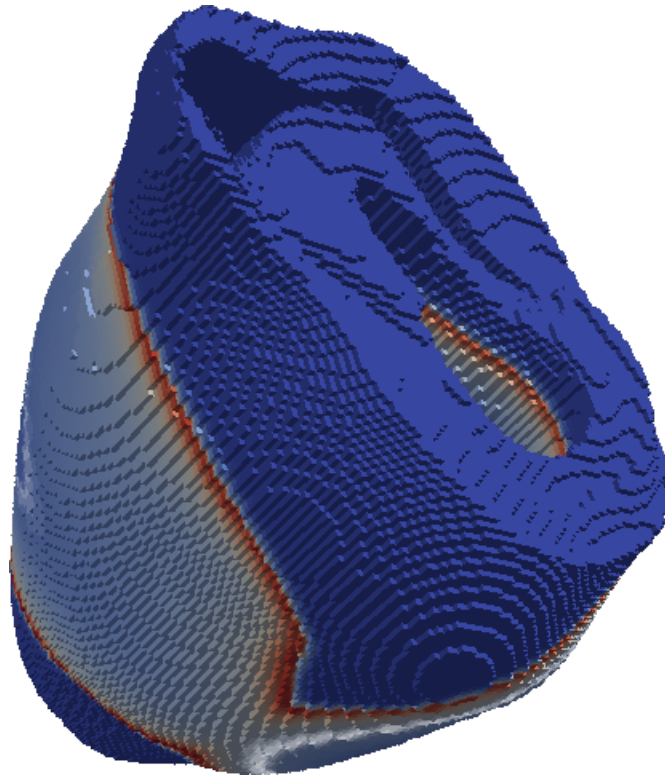


Figura 5.1. Simulação de propagação da onda elétrica (distribuição do potencial transmembrânico V_m) em tecido computacional tridimensional do coração de um coelho utilizando o esquema de adaptatividade do ALG.

5.2.1 Paralelização das operações da malha ALG

Neste trabalho, não paralelizamos nenhuma operação relativa ao refinamento e desrefinamento da malha ALG. Acreditamos que com essa abordagem podemos acelerar ainda mais as simulações.

5.2.2 Implementação da malha adaptativa ALG utilizando *clusters* de GPUs

Dada a ubiquidade de GPUs em clusters de computadores, a implementação proposta na seção acima poderia ser adaptada para a utilização de múltiplas placas gráficas. Porém, o desbalanceamento de carga gerado pela natureza adaptativa do ALG também terá que ser tratada. A nossa proposta é criar um algoritmo para tratar dinamicamente todos os problemas de balanceamento de carga gerados pelo uso de adaptatividade. Esse trabalho já foi iniciado e publicado em Barros et al. [2012], onde foi desenvolvido um modelo de eletrofisiologia cardíaca que captura a microestrutura do tecido cardíaco, utilizando uma discretização espacial muito fina ($8\mu m$). Para lidar com os

desafios computacionais, o modelo foi paralelizado usando uma abordagem híbrida: a computação em *cluster* e GPGPU. Essa implementação paralela foi capaz de reduzir os tempos de execução das simulações de mais de 6 dias em um único processador para 21 minutos em um cluster pequeno com 8 nós e equipado com 16 GPUs, i.e. 2 GPUs por nó.

5.2.3 Utilização de preconditionadores

Como mostrado na seção 4.2.6.1, tabela 4.5, o número de iterações executadas pelo método do gradiente conjugado aumenta quando utilizamos a malha adaptativa ALG. Sendo assim, poderíamos lançar mão de um preconditionador, por exemplo Jacobi, para reduzir esse número de iterações e acelerar ainda mais as simulações da eletrofisiologia cardíaca.

5.2.4 Balanceamento de carga na GPU

Com o lançamento recente de novas placas gráficas, a arquitetura CUDA agora permite um controle maior das *threads* executadas pela GPU, facilitando consideravelmente o desenvolvimento de técnicas de balanceamento de carga. Sendo assim, tencionamos adaptar o código que implementa o método de Euler adaptativo para que ele utilize as melhorias nas novas GPUs visando um melhor balanceamento de carga, que provavelmente levará a uma melhora no desempenho deste método.

5.2.5 Desenvolvimento de diferentes estratégias para a decisão de refinamento/desrefinamento no ALG

Neste trabalho, os critérios utilizados para refinamento e desrefinamento baseiam-se no fluxo através da interface de células vizinhas. Isto é, se o valor absoluto do fluxo é maior do que um limiar pré-definido de refinamento, o programa refina esta célula, enquanto que, se o valor absoluto do fluxo de todas as quatro células de um cacho é menor do que um limiar de desrefinamento, o programa desrefina o cacho. Para a nossa aplicação, os valores para os limiares de refinamento e desrefinamento foram encontrados empiricamente.

Esta solução apresenta algumas problemas difíceis de serem contornados, pois não há uma forma sistemática de encontrarem os valores dos limiares de refinamento e desrefinamento. Isto faz com que estes valores sejam definidos empiricamente, o que pode resultar em decisões equivocadas, interferindo diretamente no desempenho do algoritmo com relação ao tempo de execução e erros numéricos.

Sendo assim, acreditamos que poderíamos utilizar, por exemplo, estimadores de erros que auxiliassem na escolha desses limiares em tempo de execução, aumentando ou diminuindo esses limiares dinamicamente, de acordo com a necessidade da aplicação.

Referências Bibliográficas

- Amorim, R. M. (2009). Solução das equações do bidomínio em processadores gráficos. Dissertação de mestrado, Universidade Federal de Juiz de Fora.
- Amorim, R. M. & dos Santos, R. W. (2012). Solving the cardiac bidomain equations using graphics processing units. *Journal of Computational Science*. ISSN 1877-7503.
- Amorim, R. M.; Haase, G.; Liebmann, M. & dos Santos, R. W. (2009). Comparing CUDA and OpenGL implementations for a Jacobi iteration. Em *International Conference on High Performance Computing & Simulation (HPCS '09)*, pp. 22–32.
- Andrade, G.; Teixeira, F.; Xavier, C.; Oliveira, R.; Rocha, L. & Evsukoff, A. (2012). Hasch: High performance automatic spell checker for portuguese texts from the web. *Procedia Computer Science*, 9(0):403 – 411. ISSN 1877-0509.
- Barros, B. G.; Oliveira, R. S. ad Lobosco, M.; dos Santos, R. W. & Meira Jr., W. (2012). Simulations of Complex and Microscopic Models of Cardiac Electrophysiology Powered by Multi-GPU Platforms. *Computational and Mathematical Methods in Medicine*, pp. 1--13. ISSN 1748-670X.
- Bartocci, E.; Cherry, E. M.; Glimm, J.; Grosu, R.; Smolka, S. A. & Fenton, F. H. (2011). Toward Realtime Simulation of Cardiac Dynamics. Em *In Proc. of CMSB'11, the 9th International Conference on Computational Methods in Systems Biology*.
- Bendahmane, M.; Bürger, R. & Ruiz-Baier, R. (2010). A multiresolution space-time adaptive scheme for the bidomain model in electrocardiology. *Numerical Methods for Partial Differential Equations*, 26(6):1377--1404.
- Bondarenko, V.; Szigeti, G.; Bett, G.; Kim, S. & Rasmusson, R. (2004). Computer model of action potential of mouse ventricular myocytes. *American Journal of Physiology - Heart and Circulatory Physiology*, 287:H1378–H1403.

- Bortoli, A. L. d. (2000). *Introdução À Dinâmica De Fluidos Computacional*. Editora da Universidade. ISBN 8570255454.
- Burgarelli, D. (1998). *Modelagem computacional e simulação numérica adaptativa de equações diferenciais parciais evolutivas aplicadas a um problema termoacústico*. Tese de doutorado, Tese de doutorado, PUC-Rio, Rio de Janeiro, Brasil, 1998 (in Portuguese).
- Burgarelli, D. & Kischinhevsky, M. (1999). Efficient numerical simulation of a simplified thermoacoustic engine with new adaptive mesh refinement tools. *Computational Methods in Engineering*, 99.
- Burgarelli, D.; Kischinhevsky, M. & Biezuner, R. J. (2006). A new adaptive mesh refinement strategy for numerically solving evolutionary pde's. *J. Comput. Appl. Math.*, 196:115--131. ISSN 0377-0427.
- Burgarelli, D.; Kischinhevsky, M.; Paes Leme, P. & Silveira, O. (1995). Refinamento de malha adaptativa em paralelo,(c3ad)(colloquia em computação científica de alto desempenho). *Rio de Janeiro*.
- Butcher, J. (2008). *Numerical methods for ordinary differential equations*. Wiley.
- C. Luo & Yoram Rudy (1991). A model of the ventricular cardiac action potential. depolarization, repolarization, and their interaction. *Circ Res*, 68(6):1501--26.
- Campos, R. S.; Lobosco, M. & dos Santos, R. W. (2011). Adaptive time step for cardiac myocyte models. *Procedia Computer Science*, 4(0):1092 – 1100. ISSN 1877-0509. Proceedings of the International Conference on Computational Science, ICCS 2011.
- Catalyurek, U.; Ferreira, R.; Hartley, T.; Teodoro, G. & Sachetto, R. (2010). Data Flow Frameworks for Emerging Heterogeneous Architectures and Their Application to Biomedicine. Em *Scientific Computing with Multicore and Accelerators*, Chapman & Hall/CRC Computational Science, pp. 375--392. CRC Press.
- Cherry, E.; Greenside, H. & Henriquez, C. (2000). A space-time adaptive method for simulating complex cardiac dynamics. *Physical Review Letters*, 84(6):1343--1346.
- Cherry, E. M.; Greenside, H. S. & Henriquez, C. S. (2003). Efficient simulation of three-dimensional anisotropic cardiac tissue using an adaptive mesh refinement method. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 13(3):853--865.

- Coudiere, y.; Pierre, C. & Turpault, R. (2009). A 2d/3d finite volume method used to solve the bidomain equations of electrocardiology. Em *proceedings of Algoritmy*, pp. 1--10.
- Coutinho, B.; Teodoro, G.; Oliveira, R.; Neto, D. & Ferreira, R. (2009). Profiling general purpose gpu applications. Em *Proceedings of the 2009 21st International Symposium on Computer Architecture and High Performance Computing*, SBAC-PAD '09, pp. 11--18. ISSN 1550-6533.
- Crank, J. & Nicolson, P. (1947). A practical method for numerical evaluation of solutions of partial differential equations of the heat-conduction type. Em *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 43, pp. 50--67. Cambridge Univ Press. ISSN 1469-8064.
- D. Sato; Y. Xie; J. N. Weiss; Z. Qu; A. Garfinkel & A. R. Sanderson (2009). Acceleration of cardiac tissue simulation with graphic processing units. *Med Biol Eng Comput*, 47:1011--1015.
- Dagum, L. & Menon, R. (1998). Openmp: an industry standard api for shared-memory programming. *Computational Science Engineering, IEEE*, 5(1):46 --55. ISSN 1070-9924.
- Deuffhard, P.; Erdmann, B.; Roitzsch, R. & Lines, G. T. (2009). Adaptive finite element simulation of ventricular fibrillation dynamics. *Computing and visualization in science*, 12(5):201--205.
- Eymard, R.; Gallouët, T. & Herbin, R. (2000). Finite volume methods. *Handbook of numerical analysis*, 7:713--1018.
- Franzone, P.; Deuffhard, P.; Erdmann, B.; Lang, J. & Pavarino, L. (2007). Adaptivity in space and time for reaction-diffusion systems in electrocardiology. *SIAM journal on scientific computing*, 28(3):942.
- Gima, K. & Rudy, Y. (2002). Ionic current basis of electrocardiographic waveforms: A model study. *Circulation Research*, 90:889--896.
- GPGPU (2010). General-purpose computation on graphics processing units. Publicação Eletrônica.
- Harrild, D. & Henriquez, C. (1997). A finite volume model of cardiac propagation. *Annals of biomedical engineering*, 25(2):315--334. ISSN 0090-6964.

- Henriquez, C. (1993). Simulating the electrical behavior of cardiac tissue using the bidomain model. *Critical reviews in biomedical engineering*, 21(1):1. ISSN 0278-940X.
- Hestenes, M. & Stiefel, E. (1952). Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards Vol*, 49(6).
- Hille, B. (2001). *Ionic Channels of Excitable Membranes*. Sinauer Associates, 3 edição.
- 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.
- Hunter, P. & Borg, T. (2003). Integration from proteins to organs: the physiome project. *Nature Reviews Molecular Cell Biology*, 4(3):237--243.
- Iyer, V.; Mazhari, R. & Winslow, R. (2004). A computational model of the human left-ventricular epicardial myocyte. *Biophysical journal*, 87(3):1507--1525. ISSN 0006-3495.
- Keener, J. & Sneyd, J. (1998). *Mathematical Physiology*. Springer.
- Kumar, V. (2002). *Introduction to Parallel Computing*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. ISBN 0201648652.
- LeVeque, R. (2007). *Finite difference methods for ordinary and partial differential equations*. Siam. ISBN 0898716292.
- MacLachlan, M.; Sundnes, J. & Spiteri, R. (2007). A comparison of non-standard solvers for odes describing cellular reactions in the heart. *Computer Methods in Biomechanics and Biomedical Engineering*, 10(5):317--326.
- Miller, W. & Geselowitz, D. (1978). Simulation studies of the electrocardiogram. I. the normal heart. *Circ Res*, 43(2):301–315.
- N. Bell & M. Garland (2008). Efficient Sparse Matrix-Vector Multiplication on CUDA. Relatório técnico, NVidia Corporation.
- Netter, F. H. (2008). *Atlas de Anatomia Humana*. Elsevier Brazil.
- Oliveira, R. S. (2008). Ajuste automático de modelos celulares apoiado por algoritmos genéticos. Dissertação de mestrado, Universidade Federal de Juiz de Fora.

- Oliveira, R. S.; Rocha, B.; Amorim, R.; Campos, F.; Meira Júnior, W.; Toledo, E. & dos Santos, R. (2011). Comparing cuda, opencl and opengl implementations of the cardiac monodomain equations. Em *Proceedings of the 9th international conference on Parallel processing and applied mathematics: Part I*. Springer-Verlag.
- Oliveira, R. S.; Rocha, B. M.; Burgarelli, D.; Meira Jr., W. & dos Santos, R. W. (2012a). An adaptive mesh algorithm for the numerical solution of electrical models of the heart. Em *Proceedings of the 12th international conference on Computational Science and Its Applications - Volume Part I, ICCSA'12*, pp. 649--664, Berlin, Heidelberg. Springer-Verlag.
- Oliveira, R. S.; Rocha, B. M.; Burgarelli, D.; Meira Jr., W. & dos Santos, R. W. (2012b). A parallel accelerated adaptive mesh algorithm for the solution of electrical models of the heart. *International Journal of High Performance Systems Architecture*.
- OMS (2010). Organização Mundial da Saúde. Publicação Eletrônica - <http://www.who.int/>. Último acesso em 17 de Março de 2010.
- Owens, J.; Luebke, D.; Govindaraju, N.; Harris, M.; Krüger, J.; Lefohn, A. & Purcell, T. (2007). A survey of general-purpose computation on graphics hardware. Em *Computer graphics forum*, volume 26, pp. 80--113. Wiley Online Library.
- Pain, C.; Umpleby, A.; De Oliveira, C. & Goddard, A. (2001). Tetrahedral mesh optimisation and adaptivity for steady-state and transient finite element calculations. *Computer Methods in Applied Mechanics and Engineering*, 190(29):3771--3796.
- Plank, G.; Burton, R.; Hales, P.; Bishop, M.; Mansoori, T.; Bernabeu, M.; Garny, A.; Prassl, A.; Bollensdorff, C.; Mason, F. et al. (2009). Generation of histo-anatomically representative models of the individual heart: tools and application. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 367(1896):2257--2292.
- Plank, G.; Liebmann, M.; Weber dos Santos, R.; Vigmond, E. & Haase, G. (2007). Algebraic Multigrid Preconditioner for the Cardiac Bidomain Model. *IEEE Trans Biomed Eng*, 54:585--96.
- 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. ISBN 0071232656.
- Rocha, B. M.; Campos, F. O.; Amorim, R. M.; Plank, G.; dos Santos, R. W.; Liebmann, M. & Haase, G. (2010). Accelerating cardiac excitation spread simulations using graphics processing units. *Concurrency and Computation: Practice and Experience*.
- Roth, B. (1997). Electrical conductivity values used with the bidomain model of cardiac tissue. *Biomedical Engineering, IEEE Transactions on*, 44(4):326--328. ISSN 0018-9294.
- Rush, S. & Larsen, H. (1978). A practical algorithm for solving dynamic membrane equations. *Biomedical Engineering, IEEE Transactions on*, (4):389--392.
- Ryoo, S.; Rodrigues, C.; Bagsorkhi, S.; Stone, S.; Kirk, D. & Hwu, W. (2008). Optimization principles and application performance evaluation of a multithreaded gpu using cuda. Em *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming*, pp. 73--82. ACM.
- Sachse, F. B. (2004). *Computational Cardiology: Modeling of Anatomy, Electrophysiology and Mechanics*. Springer.
- Sagan, H. (1994). *Space-filling curves*, volume 2. Springer-Verlag New York.
- Shewchuk, J. (1994). An introduction to the conjugate gradient method without the agonizing pain.
- Southern, J.; Gorman, G.; Piggott, M. & Farrell, P. (2012). Parallel anisotropic mesh adaptivity with dynamic load balancing for cardiac electrophysiology. *Journal of Computational Science*, 3:8--16.
- Southern, J.; Gorman, G.; Piggott, M.; Farrell, P.; Bernabeu, M. & Pitt-Francis, J. (2010). Simulating cardiac electrophysiology using anisotropic mesh adaptivity. *Journal of Computational Science*, 1(2):82 -- 88. ISSN 1877-7503.
- Street, A. & Plonsey, R. (1999). Propagation in cardiac tissue adjacent to connective tissue: two-dimensional modeling studies. *Biomedical Engineering, IEEE Transactions on*, 46(1):19--25.
- Strikwerda, J. (2004). *Finite difference schemes and partial differential equations*. Society for Industrial Mathematics. ISBN 0898715679.

- Sundnes, J. (2006). *Computing the electrical activity in the heart*. Springer Verlag. ISBN 3540334327.
- Sundnes, J.; Lines, G.; Grottum, P. & Tveito, A. (2003a). Numerical Methods and Software for Modeling the Electrical Activity in the Human Heart. *Advanced Topics in Computational Partial Differential Equations–Numerical Methods and Diffpack Programming*. Springer.
- Sundnes, J.; Lines, G. T. & Tveito, A. (2003b). Ode-solvers for a stiff system arising in the modeling of the electrical activity of the heart. *International Journal of Nonlinear Sciences and Numerical Simulation*, 4(1):67--80.
- Sundnes, J.; Nielsen, B.; Mardal, K.; Cai, X.; Lines, G. & Tveito, A. (2006). On the computational complexity of the bidomain and the monodomain models of electrophysiology. *Annals of biomedical engineering*, 34(7):1088--1097. ISSN 0090-6964.
- Ten Tusscher, K.; Noble, D.; Noble, P. J. & Panfilov, A. V. (2004). A model for human ventricular tissue. *American Journal of Physiology- Heart and Circulatory Physiology*, 286(4):1573--1589.
- Teodoro, G.; Sachetto, R.; Fireman, D.; Guedes, D. & Ferreira, R. (2009a). Exploiting computational resources in distributed heterogeneous platforms. Em *Proceedings of the 2009 21st International Symposium on Computer Architecture and High Performance Computing*, SBAC-PAD '09, pp. 83--90, Washington, DC, USA. IEEE Computer Society.
- Teodoro, G.; Sachetto, R.; Sertel, O.; Gurcan, M.; Meira, W.; Catalyurek, U. & Ferreira, R. (2009b). Coordinating the use of gpu and cpu for improving performance of compute intensive applications. Em *Cluster Computing and Workshops, 2009. CLUSTER '09. IEEE International Conference on*, pp. 1 --10. ISSN 1552-5244.
- Trangenstein, J. & Kim, C. (2004). Operator splitting and adaptive mesh refinement for the Luo-Rudy I model. *Journal of Computational Physics*, 196(2):645--679.
- Tung, L. (1978). *A bi-domain model for describing ischemic myocardial dc potentials*. Tese de doutorado, Massachusetts Institute of Technology.
- Vigmond, E.; Weber dos Santos, R.; Prassl, A.; Deo, M. & Plank, G. (2008). Solvers for the cardiac bidomain equations. *Progress in biophysics and molecular biology*, 96(1-3):3--18. ISSN 0079-6107.

- Vigmond, E. J.; Aguel, F. & Trayanova, N. (2002). Computational techniques for solving the bidomain equations in three dimensions. *IEEE Trans Biomed Eng*, 49(11):1260–9.
- Weber dos Santos, R.; Plank, G.; Bauer, S. & Vigmond, E. J. (2004a). Parallel multigrid preconditioner for the cardiac bidomain model. *IEEE Trans Biomed Eng*, 51(11):1960–8.
- Weber dos Santos, R.; Plank, G.; Bauer, S. & Vigmond, E. J. (2004b). Preconditioning Techniques for the Bidomain Equations. *Lecture Notes In Computational Science And Engineering*, 40:571–580.
- Weiser, M.; Erdmann, B. & Deuffhard, P. (2010). On efficiency and accuracy in cardioelectric simulation. Em *Progress in Industrial Mathematics at ECMI 2008*, pp. 371--376. Springer.
- Xavier, C. R.; Oliveira, R. S.; da Fonseca Vieira, V.; dos Santos, R. W. & Meira Jr., W. (2009). Multi-level parallelism for the cardiac bidomain equations. *International Journal of Parallel Programming*, 37:572–592. ISSN 0885-7458.
- Y. Saad (1996). *Iterative Methods for Sparse Linear Systems*. PWS Publishing Company.

Apêndice A

Modelagem da eletrofisiologia cardíaca

A.1 Potenciais de Equilíbrio da Membrana Celular

A descrição da membrana celular como um circuito resistor-capacitor não considera a existência de uma diferença de potencial elétrico através da mesma [Sachse, 2004]. Essa diferença é encontrada devido à diferença de concentração de íons através da membrana e a diferença de permeabilidade da membrana aos diversos íons. As equações de Nernst e Goldman-Hodgin-Katz são usadas na modelagem da eletrofisiologia e descrevem esta diferença de potencial.

A.1.1 O Potencial de Nernst

Nesta seção iremos mostrar a equação de Nernst, a qual descreve o potencial de equilíbrio $\phi_i - \phi_e$ através da membrana celular resultante das concentrações iônicas $[k]_i$ e $[k]_e$. No equilíbrio o fluxo do íon k devido a forças elétricas $j_{E,k}$ e a difusão $j_{D,k}$ é igual a zero (veja figura A.1).

O fluxo de íons através dos canais existentes na membrana irá depender do potencial transmembrânico. Para uma corrente puramente resistiva um bom modelo seria:

$$\vec{J} = \sigma \vec{E} \quad (\text{A.1})$$

Aqui temos a lei de Ohm, na qual $\vec{J}$ é a corrente, $\vec{E}$ é o campo elétrico e σ é uma condutividade constante. Porém, os canais iônicos são mais complicados do que simples resistências [Sundnes et al., 2003a]. O fluxo depende da concentração iônica e

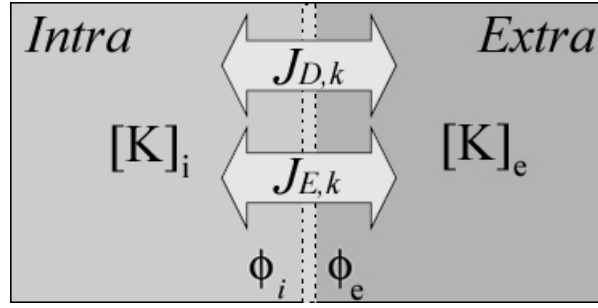


Figura A.1. Fluxo, potenciais e concentrações iônicas da equação de Nernst (adaptada de [Sachse, 2004]).

do potencial elétrico. Sabemos que o campo elétrico pode ser escrito como o gradiente de um potencial escalar, $\vec{E} = -\nabla\phi$. Se assumirmos que não existem gradientes de concentração, o fluxo de íons $j_{E,k}$ gerado por um campo elétrico é dado pela equação de

Planck [Sachse, 2004]:

$$\vec{j}_{E,k} = -u_k \frac{z_k}{|z_k|} [k] \nabla\phi, \quad (\text{A.2})$$

em que u_k é a mobilidade do íon k , z_k é a valência do íon, $[k]$ é a concentração e ϕ é o potencial elétrico.

Além do fluxo devido ao campo elétrico, existe também um fluxo gerado pelo gradiente de concentração. Em um campo elétrico neutro esse fluxo não será zero pois os íons se moverão para regiões de menor concentração. Isto pode ser modelado pela lei de Fick [Sachse, 2004]:

$$\vec{j}_{D,k} = -D_k \nabla[k], \quad (\text{A.3})$$

onde D_k é o coeficiente de difusão do íon k . No caso geral temos tanto campos elétricos não-nulos quanto gradientes de concentração. Por esse motivo, a corrente total é a soma da corrente difusiva com a corrente gerada pelo campo elétrico,

$$\vec{j}_k = \vec{j}_{D,k} + \vec{j}_{E,k} \quad (\text{A.4})$$

Existe uma relação, determinada por Einstein [Keener & Sneyd, 1998], entre a mobilidade u_k e a difusão de Fick D :

$$u_k = D \frac{|z_k|}{RT}, \quad (\text{A.5})$$

em que F é a constante de Faraday, R é a constante universal dos gases perfeitos e T

é a temperatura absoluta. Assim a corrente total pode ser escrita como:

$$\vec{j}_k = -D \left(\nabla[k] + \frac{z_k F}{RT} [k] \nabla \phi \right) \quad (\text{A.6})$$

Nesse sistema o equilíbrio é alcançado quando o fluxo total j_k do íon k através da membrana é zero. Assim o equilíbrio é alcançado se

$$\vec{j}_k = j_{D,k} + j_{E,k} = 0 \quad (\text{A.7})$$

Para o fluxo através dos canais na membrana é razoável considerar somente variações ao longo do comprimento do canal [Keener & Sneyd, 1998]. Podemos também ajustar o sistema de coordenadas tal que o eixo x esteja ao longo do comprimento do canal, com $x = 0$ sendo a fronteira interior do canal e $x = L$ a fronteira exterior. Em 1D, com $\vec{j}_k = 0$, a equação (A.6) ficaria:

$$\frac{d[k]}{dx} + \frac{z_k F}{RT} [k] \frac{d\phi}{dx} = 0. \quad (\text{A.8})$$

Dividindo por $[k]$ e integrando de 0 até L temos

$$\int_0^L \frac{1}{[k]} \frac{d[k]}{dx} dx + \int_0^L \frac{z_k F}{RT} \frac{d\phi}{dx} dx = 0 \quad (\text{A.9})$$

e finalmente

$$\ln([k]) \Big|_{[k](0)}^{[k](L)} = -\frac{z_k F}{RT} (\phi(L) - \phi(0)) = \frac{z_k F}{RT} E_k, \quad (\text{A.10})$$

sendo que $E_k = \phi_i - \phi_e$. O valor do potencial transmembrânico com fluxo zero é

$$E_k = \frac{RT}{z_k F} \ln \left(\frac{[k]_e}{[k]_i} \right), \quad (\text{A.11})$$

onde $[k]_e$ e $[k]_i$ são as concentrações de k fora e dentro da célula, respectivamente. O potencial E_k , para qual o fluxo é zero, é denominado potencial de equilíbrio de Nernst. Podemos notar que a equação de Nernst só é válida quando consideramos a existência de um único tipo de íon.

A.1.2 Equação de Goldman-Hodgkin-Katz

A equação de Goldman-Hodgkin-Katz (GHK) foi desenvolvida para descrever o potencial de equilíbrio $\phi_i - \phi_e$ através da membrana celular resultante de diferentes concentrações iônicas de íons distintos, como potássio, sódio e cloro. Essa equação estende a

equação de Nernst para o caso de mais de um tipo de íon. A concentração de cada tipo de íon é determinada para os espaços intra- e extracelular. Além disso, existem os fluxos de cada íon causados tanto pela difusão quanto pelas forças elétricas. A figura A.2 ilustra a situação quando temos potássio, sódio e cloro.

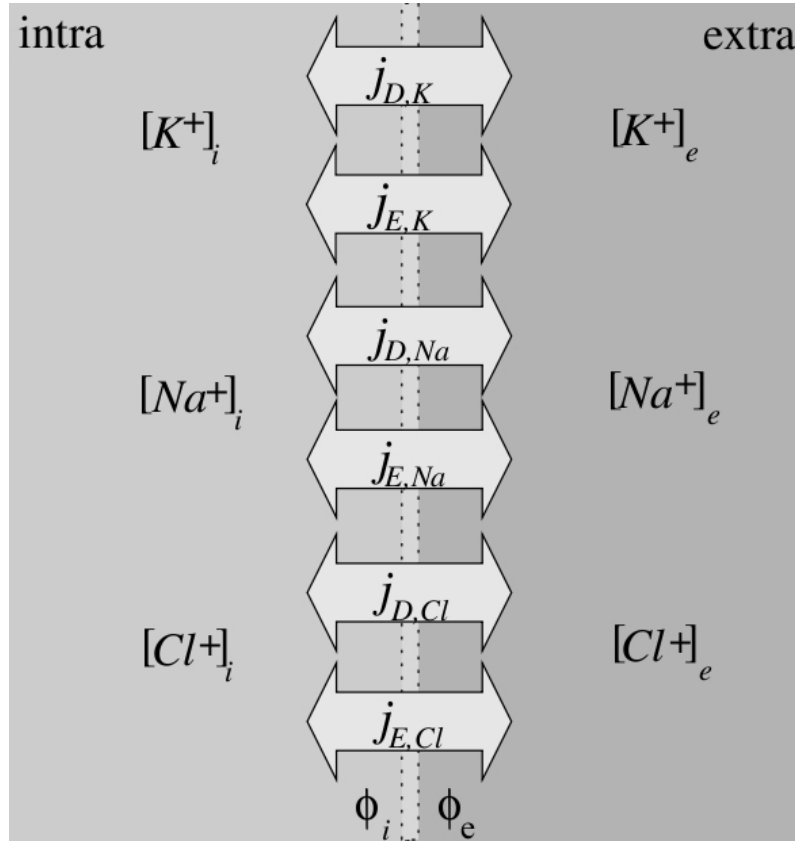


Figura A.2. Fluxo, potenciais e concentrações iônicas da equação de Goldman-Hodgkin-Katz (adaptada de [Sachse, 2004]).

Para esta situação, a equação de Goldman-Hodgkin-Katz determina o potencial de equilíbrio ou de repouso E_r da forma [Keener & Sneyd, 1998]:

$$E_r = -\frac{RT}{F} \ln \frac{P_K[K^+]_i + P_{Na}[Na^+]_i + P_{Cl}[Cl^-]_e}{P_K[K^+]_e + P_{Na}[Na^+]_e + P_{Cl}[Cl^-]_i} \quad (A.12)$$

a partir das concentrações iônicas, das permeabilidades da membrana a determinados íons e da temperatura absoluta T . A permeabilidade da membrana para o potássio, sódio e cloro é representada por P_K , P_{Na} e P_{Cl} respectivamente. A permeabilidade de um íon k é expressa por [Sachse, 2004]:

$$P_k = \frac{D_k \beta_k}{h} \quad (A.13)$$

sendo k a espessura da membrana, D_k o coeficiente de difusão e β_k o coeficiente de partição água-membrana. Os coeficientes de difusão e de partição água membrana são dependentes do tipo de membrana e do íon k .

Para deduzirmos a equação GHK precisamos fazer algumas hipóteses simplificadoras. Considera-se que a membrana é homogênea, plana e infinita. Presume-se também que as concentrações intra e extracelulares são homogêneas, que o campo elétrico na membrana é constante e que as correntes iônicas são independentes entre si [Hille, 2001].

A.2 Modelos para a Corrente Iônica

Quando o potencial transmembrânico é diferente do potencial de equilíbrio de Nernst, uma corrente de íons passa através do canal. A forma mais simples de expressar a corrente iônica e satisfazer o princípio de Nernst é por meio de uma formulação linear [Sundnes et al., 2003a]:

$$I_k = g_k(V_m - E_k), \quad (\text{A.14})$$

com g_k a condutividade do íon e E_k o potencial de equilíbrio de Nernst do íon k .

Podemos derivar outro modelo para a corrente iônica se assumirmos que o campo elétrico é constante. Seguindo a notação introduzida em A.8, podemos escrever que $\nabla\phi = v/L$, em que v é o potencial constante e L é o comprimento do canal. Considerando novamente o caso em 1 dimensão obtemos:

$$\frac{d[k]}{dx} - \frac{z_k F V_m}{RT L} [k] + \frac{j_k}{D} = 0 \quad (\text{A.15})$$

Essa é uma equação diferencial ordinária em $[k]$ com os valores nos pontos extremos conhecidos, e j_k uma incógnita a ser determinada. Resolvendo a equação obtemos a seguinte expressão para o fluxo,

$$j_k = \frac{D}{L} \frac{z_k F V_m}{RT} \frac{[k]_i - [k]_e \exp\left(\frac{-z_k F E_k}{RT}\right)}{1 - \exp\left(\frac{-z_k F V_m}{RT}\right)}. \quad (\text{A.16})$$

Apesar dessa expressão ser bem mais complicada que a expressão linear, é fácil verificar que o fluxo é zero quando $V_m = E_k$.

A.3 Canais Iônicos

Até agora descrevemos o comportamento de canais iônicos com condutividades constantes. Porém, a condutividade dos canais iônicos pode mudar ao longo do tempo em resposta a mudanças no potencial transmembrânico. Isso ocorre porque as flutuações do potencial transmembrânico influenciam as partes carregadas das proteínas que compõem os canais iônicos. Por sua vez, estas podem se mover e alterar a estrutura do canal.

A.3.1 O Modelo de Dois Estados

O comportamento de um canal iônico isolado pode ser modelado por estados e funções que descrevem a transição entre esses estados. No caso mais simples somente dois estados são levados em consideração: aberto e fechado. A transição entre os estados é estocástica sendo O_i a probabilidade do canal estar aberto e C_i é a probabilidade dele estar fechado. Temos também que $O_i + C_i = 1$. Além disso O_i e $C_i \in [0, 1]$.

Fazendo $O_i = n$, a variação da probabilidade do canal estar aberto O é determinada por:

$$\frac{dn}{dt} = \alpha(V_m)(1 - n) - \beta(V_m)n \quad (\text{A.17})$$

sendo $\alpha(V_m)$ uma taxa responsável pela transição do estado fechado para o estado aberto: $C_i \Rightarrow O_i$; e $\beta(V_m)$ a taxa responsável pela transição do estado aberto para o estado fechado: $O_i \Rightarrow C_i$. As taxas $\alpha(V_m)$ e $\beta(V_m)$ dependem do tipo de canal iônico, potencial transmembrânico, concentração iônica, entre outros fatores. No equilíbrio a variação é zero:

$$\frac{dn}{dt} = 0 \quad (\text{A.18})$$

A Equação A.17 pode ser convenientemente reescrita na forma:

$$\frac{dn}{dt} = \frac{n_\infty(V_m) - n}{\tau_n(V_m)} \quad (\text{A.19})$$

onde

$$n_\infty(V_m) = \frac{\alpha(V_m)}{\alpha(V_m) + \beta(V_m)} \quad (\text{A.20})$$

é o valor de equilíbrio assintótico de n , e

$$\tau_n(V_m) = \frac{1}{\alpha(V_m) + \beta(V_m)} \quad (\text{A.21})$$

é a constante de tempo de n .

Expressões para $n_\infty(V_m)$ e $\tau_n(V_m)$ podem ser obtidas diretamente de dados experimentais [Keener & Sneyd, 1998].

A condutividade macroscópica de uma população de canais similares é dada por:

$$g_i = N_i \cdot n \cdot g_{i,max} \quad (\text{A.22})$$

com N_i representando o número de canais e $g_{i,max}$ a condutividade máxima do canal.

A.3.2 O Modelo de Subunidades

No modelo de dois estados, o canal possuía somente uma unidade que poderia estar aberta ou fechada. Porém para modelarmos canais iônicos mais complexos temos que considerar que o canal pode ser formado por diferentes subunidades. Essas subunidades são independentes e podem estar abertas ou fechadas. Para ilustrar iremos mostrar a modelagem de um canal de sódio hipotético.

Os canais de sódio exibem um rápido aumento da condutividade em resposta às variações no potencial transmembrânico (figura A.3). Esse processo, denominado de ativação, é imediatamente seguido por um segundo processo que lentamente dirige a condutividade para zero (inativação). Para descrever o comportamento desses canais são necessários modelos que considerem ambas ativação e inativação do canal.

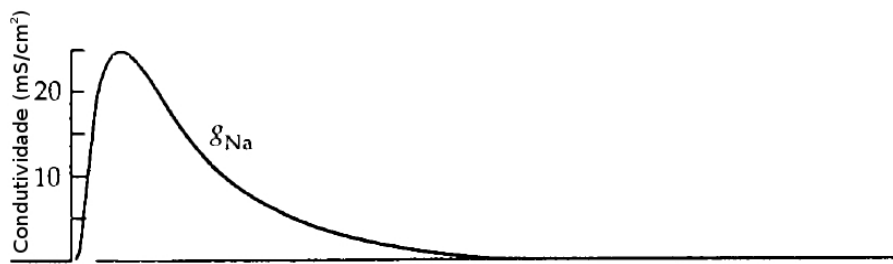


Figura A.3. Variação da condutividade de sódio (G_{Na}) em função do tempo após uma mudança no potencial transmembrânico (Adaptado de [Keener & Sneyd, 1998]).

Assumimos que o canal é formado por 3 subunidades, uma h e duas m . A equação (A.23) pode ser estendida para o caso particular de canais iônicos com duas subunidades idênticas m relacionadas a ativação e uma subunidade h associada à ina-

tivação:

$$I_{Na} = m^2 h g_{max} (V_m - E_{Na}) \quad (\text{A.23})$$

$$\frac{dm}{dt} = \frac{m_\infty(V_m) - m}{\tau_m(V_m)} \quad (\text{A.24})$$

$$\frac{dh}{dt} = \frac{h_\infty(V_m) - h}{\tau_h(V_m)} \quad (\text{A.25})$$

em que as subunidades m e h são independentes e podem estar cada uma no estado aberto ou fechado e E_{Na} é o potencial de Nernst do íon sódio.

As condições iniciais da variável de ativação m e da variável de inativação h são 0 e 1, respectivamente. Sendo assim, inicialmente $I_{Na} = 0$. Depois que V_m se afasta do potencial de Nernst (E_{Na}) m tende a $m_\infty(V_m)$, como pode ser visto pela Equação A.24, onde $m_\infty(V_m) > 0$ e tende a 1 à medida que v aumenta. Como $\tau_m(V_m)$ é uma constante de tempo muito rápida I_{Na} tende a $g_{max}(V_m - E_{Na})$ rapidamente. Em paralelo com este processo de ativação está ocorrendo o processo de inativação, ou seja, h está passando de 1 para 0, porém, com uma velocidade bem mais lenta pois $\tau_h(V_m) > \tau_m(V_m)$. Como conclusão deste comportamento temos a variação da condutividade do canal de sódio, que pode ser vista na figura A.3. Modelos mais complexos de canais iônicos podem ser encontrados em Keener & Sneyd [1998] e Hille [2001].

A.4 O Modelo de Hodgkin e Huxley

O modelo de Hodgkin e Huxley descreve a eletrofisiologia da membrana do axônio gigante de lula e foi desenvolvido a partir de medidas do comportamento elétrico passivo e ativo da célula [Hodgkin & Huxley, 1952]. A base da descrição do potencial de ação proposto por Hodgkin e Huxley é o comportamento dos canais de sódio e de potássio.

O PA pode ser dividido em três fases sucessivas. A fase de repouso, na qual diz-se que a membrana está polarizada, o potencial transmembrânico é igual ao potencial de equilíbrio. Na fase de despolarização, a membrana subitamente se torna muito permeável ao sódio, permitindo assim que um grande número de íons Na^+ disponíveis no meio extracelular se difunda no sentido do gradiente de concentração. Em grandes fibras nervosas, esse largo fluxo de íons Na^+ para o interior da célula faz com que o potencial na membrana se torne positivo. Finalmente, os canais de sódio começam a se inativar, ao passo que os canais de potássio começam a se abrir mais que o normal. A rápida difusão de potássio em direção ao meio extracelular restabelece o potencial de

equilíbrio da membrana. Essa fase é denominada repolarização. O fluxo de K^+ através dos canais iônicos pode diminuir o potencial para valores menores que o de repouso. Nesse caso, a membrana é dita hiperpolarizada. Esta relação entre potencial de ação e as variações das condutividades de sódio e potássio pode ser observada nas figuras A.4 e A.5.

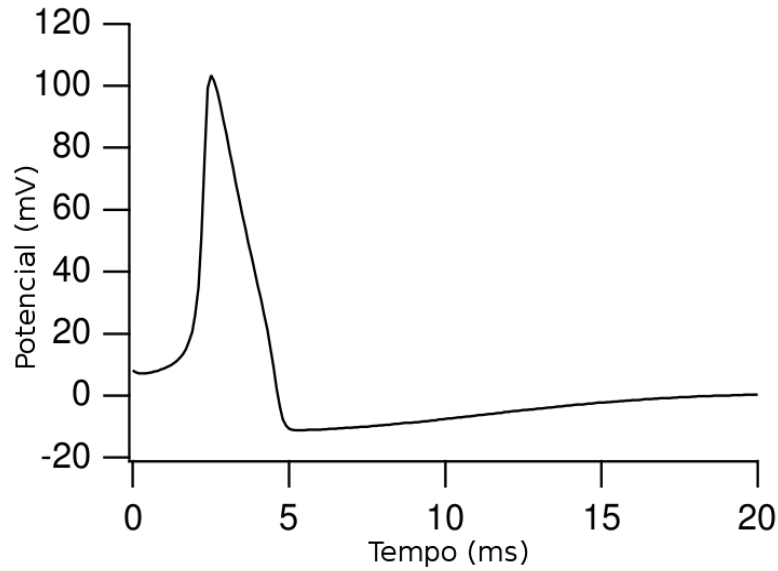


Figura A.4. Potencial de ação do modelo de Hodgkin & Huxley [1952] (adaptado de Keener & Sneyd [1998]).

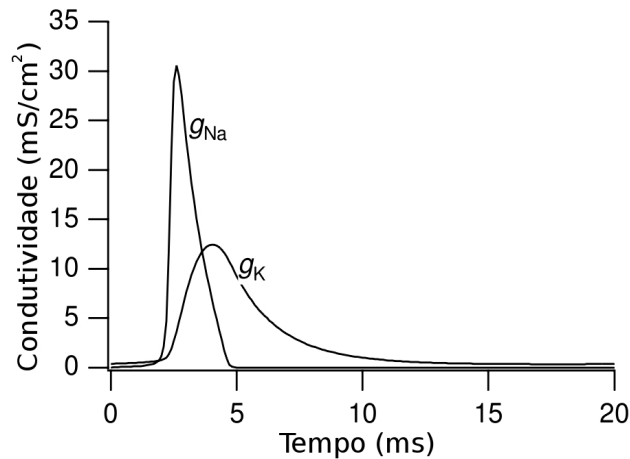


Figura A.5. Variação das condutividades g_{Na} e g_K do modelo Hodgkin-Huxley durante um potencial de ação (adaptado de Keener & Sneyd [1998]).

Usando a formulação matemática podemos calcular correntes de diferentes íons que passam através da membrana do axônio e o potencial transmembrânico. Este

potencial, denominado V_m é definido como sendo o potencial intracelular menos o potencial extracelular e a derivada no tempo de V_m é expressa por:

$$\frac{dV_m}{dt} = -\frac{1}{C_m}(I_m + I_{stim}) \quad (\text{A.26})$$

onde C_m é a capacitância da membrana, I_m a corrente transmembrânica e I_{stim} é uma corrente de estímulo. A corrente transmembrânica do modelo Hodgkin-Huxley é dada por

$$I_m = I_{Na} + I_K + I_l \quad (\text{A.27})$$

sendo I_{Na} a corrente de sódio, I_K a corrente de potássio e I_l uma corrente de fuga. A corrente de fuga I_l é uma soma de diferentes correntes iônicas, principalmente cloro. As correntes são determinadas pelas condutividades g_{Na} , g_K e g_l , respectivamente, e também pela diferenças entre o potencial transmembrânico e os potenciais de equilíbrio E_{Na} , E_K e E_l :

$$I_{Na} = g_{Na}(V_m - E_{Na}) \quad (\text{A.28})$$

$$I_K = g_K(V_m - E_K) \quad (\text{A.29})$$

$$I_l = g_l(V_m - E_l) \quad (\text{A.30})$$

Assume-se que a condutividade g_l é constante e as outras condutividades variam com tempo e são dependentes do potencial. As concentrações iônicas são consideradas constantes, o que leva a potenciais de equilíbrio também constantes.

A condutividade de sódio g_{Na} é dependente do tempo e do potencial:

$$g_{Na} = m^3 h \bar{g}_{Na} \quad (\text{A.31})$$

onde $\bar{g}_{Na}$ é a condutividade máxima de sódio, m é uma variável adimensional de ativação e h uma variável adimensional de inativação. As taxas dependentes do potencial α_m , β_m , α_h e β_h controlam as variáveis de ativação e inativação:

$$\frac{dm}{dt} = \alpha_m(1 - m) - \beta_m m \quad (\text{A.32})$$

$$\frac{dh}{dt} = \alpha_h(1 - h) - \beta_h h \quad (\text{A.33})$$

A condutividade de potássio g_k também é dependente do potencial e do tempo:

$$g_k = \bar{g}_K n^4 \quad (\text{A.34})$$

onde $\bar{g}_K$ representa a condutividade máxima de potássio e n é uma variável de estado adimensional controlada pelas taxas dependentes de potencial α_n e β_n :

$$\frac{dn}{dt} = \alpha_n(1 - n) - \beta_n n \quad (\text{A.35})$$

As funções específicas α e β propostas pro Hodgkin e Huxley são, em $(ms)^{-1}$:

$$\alpha_m = 0.1 \frac{25 - V_m}{\exp\left(\frac{25 - V_m}{10}\right) - 1}, \quad (\text{A.36})$$

$$\beta_m = 4 \exp\left(\frac{-V_m}{18}\right), \quad (\text{A.37})$$

$$\alpha_h = 0.07 \exp\left(\frac{-V_m}{20}\right), \quad (\text{A.38})$$

$$\beta_h = \frac{1}{\exp\left(\frac{30 - V_m}{10}\right) + 1}, \quad (\text{A.39})$$

$$\alpha_n = 0.01 \frac{10 - V_m}{\exp\left(\frac{10 - V_m}{10}\right) - 1}, \quad (\text{A.40})$$

$$\beta_h = 0.125 \exp\left(\frac{-V_m}{80}\right). \quad (\text{A.41})$$

A.5 O Modelo Luo Rudy

O modelo para células do ventrículo de mamíferos de C. Luo & Yoram Rudy [1991] inclui 6 correntes iônicas. O potencial transmembrânico, V_m , satisfaz:

$$\frac{dV_m}{dt} = -\frac{1}{C_m}(I_{ion} + I_{stim}), \quad (\text{A.42})$$

onde C_m é a capacitância da membrana, I_{ion} é definida como

$$I_{ion} = I_{Na} + I_{si} + I_K + I_{K1} + I_{Kp} + I_b \quad (\text{A.43})$$

sendo I_{Na} a corrente de sódio rápida, I_{si} uma corrente rápida, I_K , I_{K1} e I_{Kp} correntes de potássio e I_b uma corrente de *background*. Na figura A.6 temos uma representação esquemática das correntes incluídas no modelo.

Todas correntes iônicas do modelo são controladas por variáveis de ativação e

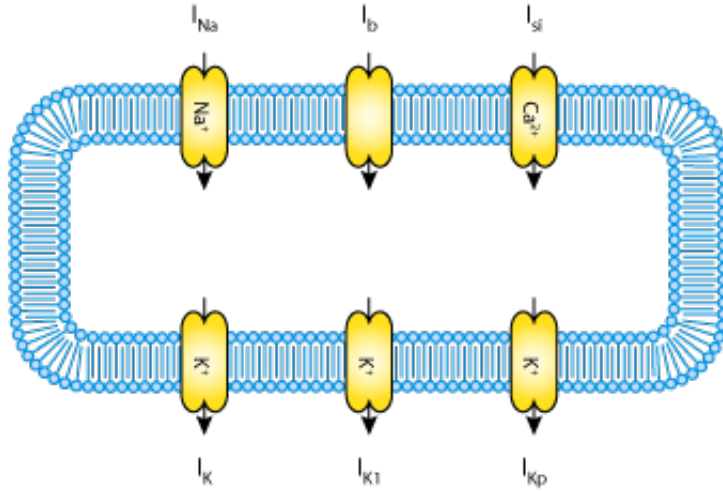


Figura A.6. Representação esquemática das correntes e bombas incluídas no modelo C. Luo & Yoram Rudy [1991].

inativação descritas por equações diferenciais ordinárias (EDOs) da forma

$$\frac{dy}{dt} = \frac{y_{\infty} - y}{\tau_y} \quad (\text{A.44})$$

onde y é a variável de ativação/inativação em questão e os termos y_{∞} e τ_y são definidos como

$$y_{\infty} = \frac{\alpha_y}{\alpha_y + \beta_y}, \quad \tau_y = \frac{1}{\alpha_y + \beta_y}, \quad (\text{A.45})$$

sendo α_y e β_y funções de V_m . Além disso, α_{k1} e β_{k1} do canal I_{K1} , dependem da concentração extracelular de potássio. Expressões completas para α_y e β_y podem ser encontradas em C. Luo & Yoram Rudy [1991].

No total, existem 8 EDOs neste modelo. Uma descrição completa pode ser encontrada em C. Luo & Yoram Rudy [1991].

A.6 O Modelo de Ten Tusscher

O modelo para células do ventrículo de humano de Ten Tusscher et al. [2004] inclui 11 correntes iônicas além de 2 correntes de trocadores e bombas. O potencial transmembrânico, V_m , satisfaz:

$$\frac{dV_m}{dt} = -\frac{1}{C_m}(I_{ion} + I_{stim}), \quad (\text{A.46})$$

onde C_m é a capacitância da membrana, I_{ion} é definida como

$$I_{ion} = I_{Na} + I_{K1} + I_{to} + I_{Kr} + I_{Ks} + I_{CaL} + I_{NaCa} + I_{NaK} + I_{pCa} + I_{pK} + I_{bCa} + I_{bNa} \quad (A.47)$$

sendo I_{NaCa} a corrente do trocador Na^+/Ca^{2+} , I_{NaK} é a corrente da bomba Na^+/K^+ , I_{pCa} e I_{pK} são corrente de *plateau* de cálcio e potássio, I_{bCa} e I_{bK} correntes de *background* de cálcio e potássio. Na figura A.7 temos uma representação esquemática das correntes incluídas no modelo.

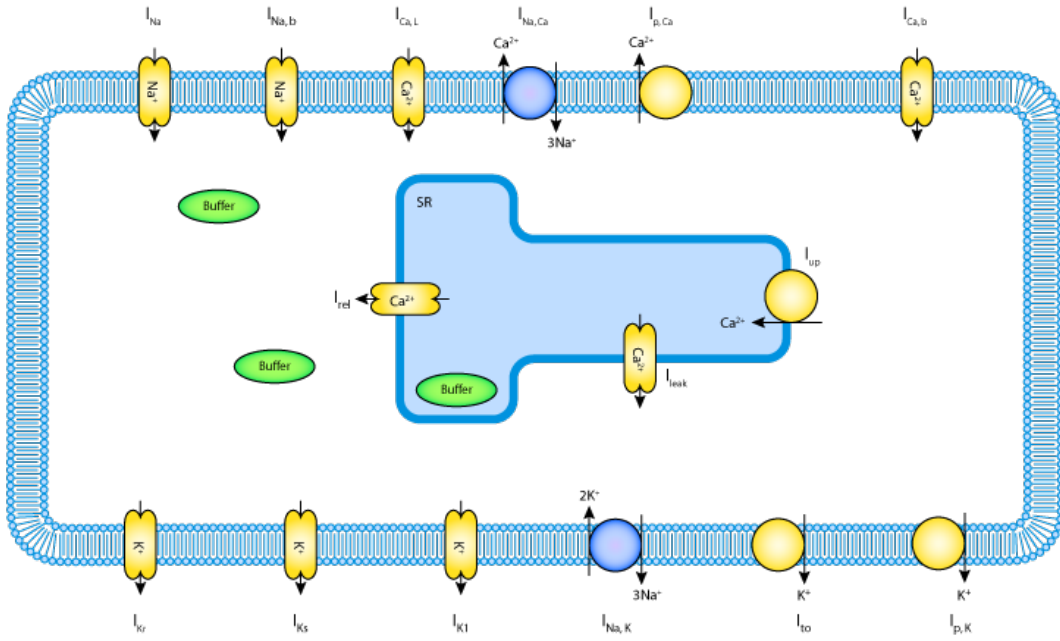


Figura A.7. Representação esquemática das correntes e bombas incluídas no modelo Ten Tusscher et al. [2004].

Como no modelo apresentado na seção anterior, as correntes iônicas são controladas por variáveis de ativação e inativação descritas por equações diferenciais ordinárias (EDOs) da forma

$$\frac{dy}{dt} = \frac{y_{\infty} - y}{\tau_y} \quad (A.48)$$

onde y é a variável de ativação/inativação em questão e os termos y_{∞} e τ_y são definidos como

$$y_{\infty} = \frac{\alpha_y}{\alpha_y + \beta_y}, \quad \tau_y = \frac{1}{\alpha_y + \beta_y}, \quad (A.49)$$

sendo α_y e β_y funções de V_m . Expressões completas para α_y e β_y podem ser encontradas em Ten Tusscher et al. [2004].

No total, existem 17 EDOs neste modelo. Uma descrição completa pode ser encontrada em Ten Tusscher et al. [2004].

Note que os autores dos diferentes modelos utilizam nomenclaturas distintas para as mesmas correntes iônicas. Neste trabalho decidimos manter os nomes das variáveis como publicadas.

A.7 Aplicação do MVF para a resolução do monodomínio

As derivadas parciais de V nas interfaces serão aproximadas usando o seguinte esquema de diferenças finitas, considerando discretizações uniformes no espaço ($\Delta x = \Delta y = h$):

$$\frac{\partial V_{m_{i+1/2,j}}}{\partial x} = \frac{V_{m_{i+1,j}} - V_{m_{i,j}}}{h} \quad (\text{A.50})$$

$$\frac{\partial V_{m_{i+1/2,j}}}{\partial y} = \frac{(V_{m_{i+1,j+1}} + V_{m_{i,j+1}} - V_{m_{i+1,j-1}} - V_{m_{i,j-1}})}{4h} \quad (\text{A.51})$$

$$\frac{\partial V_{m_{i,j+1/2}}}{\partial x} = \frac{(V_{m_{i+1,j+1}} + V_{m_{i+1,j}} - V_{m_{i-1,j+1}} - V_{m_{i-1,j}})}{4h} \quad (\text{A.52})$$

$$\frac{\partial V_{m_{i,j+1/2}}}{\partial y} = \frac{V_{m_{i,j+1}} - V_{m_{i,j}}}{h} \quad (\text{A.53})$$

$$\frac{\partial V_{m_{i-1/2,j}}}{\partial x} = \frac{V_{m_{i,j}} - V_{m_{i-1,j}}}{h} \quad (\text{A.54})$$

$$\frac{\partial V_{m_{i-1/2,j}}}{\partial y} = \frac{(V_{m_{i-1,j+1}} + V_{m_{i,j+1}} - V_{m_{i-1,j-1}} - V_{m_{i,j-1}})}{4h} \quad (\text{A.55})$$

$$\frac{\partial V_{m_{i,j-1/2}}}{\partial x} = \frac{(V_{m_{i+1,j-1}} + V_{m_{i+1,j}} - V_{m_{i-1,j-1}} - V_{m_{i-1,j}})}{4h} \quad (\text{A.56})$$

$$\frac{\partial V_{m_{i,j-1/2}}}{\partial y} = \frac{V_{m_{i,j}} - V_{m_{i,j-1}}}{h} \quad (\text{A.57})$$

Rearranjando e substituindo as discretizações na equação (2.39) temos:

$$C_m \frac{V_{m_{i,j}}^{n+1} - V_{m_{i,j}}^n}{\Delta t} = \frac{L \cdot (J_{x_{i+1/2,j}} - J_{x_{i-1/2,j}} + J_{y_{i,j+1/2}} - J_{y_{i,j-1/2}})}{\beta A} - I_{ion}(V_m, \boldsymbol{\eta}) \quad (\text{A.58})$$

Decompondo o operador diferencial na equação anterior temos:

$$C_m \frac{V_{m,i,j}^* - V_{m,i,j}^n}{\Delta t} = \frac{L \cdot (J_{x_{i+1/2,j}}^* - J_{x_{i-1/2,j}}^* + J_{y_{i,j+1/2}}^* - J_{y_{i,j-1/2}}^*)}{\beta A} \quad (\text{A.59})$$

$$C_m \frac{V_{i,j}^{n+1} - V_{m,i,j}^*}{\Delta t} = -I_{ion}(V_{m,i,j}^*, \boldsymbol{\eta}^n) \quad (\text{A.60})$$

$$\frac{\partial \boldsymbol{\eta}}{\partial t} = f(\boldsymbol{\eta}^n, V_m^*, t) \quad (\text{A.61})$$

onde $*$ é um passo intermediário e $n + 1$ é o próximo passo de tempo e,

$$J_{x_{i+1/2,j}} = \sigma_{x_{i+1/2,j}} \frac{V_{m_{i+1,j}} - V_{m_{i,j}}}{h} + \frac{\sigma_{xy_{i+1/2,j}}^1 (V_{i+1,j+1} - V_{m_{i+1,j-1}}) + \sigma_{xy_{i+1/2,j}}^2 (V_{m_{i,j+1}} - V_{m_{i,j-1}})}{4h} \quad (\text{A.62})$$

$$J_{x_{i-1/2,j}} = \sigma_{x_{i-1/2,j}} \frac{V_{m_{i,j}} - V_{m_{i-1,j}}}{h} + \frac{\sigma_{xy_{i-1/2,j}}^1 (V_{m_{i-1,j+1}} - V_{m_{i-1,j-1}}) + \sigma_{xy_{i-1/2,j}}^2 (V_{m_{i,j+1}} - V_{m_{i,j-1}})}{4h} \quad (\text{A.63})$$

$$J_{y_{i,j+1/2}} = \frac{\sigma_{xy_{i,j+1/2}}^1 (V_{m_{i+1,j+1}} - V_{m_{i-1,j+1}}) + \sigma_{xy_{i,j+1/2}}^2 (V_{m_{i+1,j}} - V_{m_{i-1,j}})}{4h} + \sigma_{y_{i,j+1/2}} \frac{V_{m_{i,j+1}} - V_{m_{i,j}}}{h} \quad (\text{A.64})$$

$$\sigma_{y_{i,j+1/2}} \frac{V_{m_{i,j+1}} - V_{m_{i,j}}}{h} \quad (\text{A.65})$$

$$J_{y_{i,j-1/2}} = \frac{\sigma_{xy_{i,j-1/2}}^1 (V_{m_{i+1,j-1}} - V_{m_{i-1,j-1}}) + \sigma_{xy_{i,j-1/2}}^2 (V_{m_{i+1,j}} - V_{m_{i-1,j}})}{4h} + \sigma_{y_{i,j-1/2}} \frac{V_{m_{i,j}} - V_{m_{i,j-1}}}{h} \quad (\text{A.66})$$

$$\sigma_{y_{i,j-1/2}} \frac{V_{m_{i,j}} - V_{m_{i,j-1}}}{h} \quad (\text{A.67})$$

A.7.1 MVF com ALG

Para exemplificarmos a aplicação do MVF com ALG utilizaremos um tecido com as fibras orientadas horizontalmente. Sendo assim teremos o seguinte esquema de dife-

renças finitas (com $\Delta x_{i,j} = \Delta y_{i,j} = h_{i,j}$):

$$\left. \frac{\partial V}{\partial x} \right|_{(i+1/2,j)} = \sum_{k=1}^{m_1} \frac{V_{d,k} - V_{i,j}}{h_1}, \quad (\text{A.68})$$

$$\left. \frac{\partial V}{\partial x} \right|_{(i-1/2,j)} = \sum_{k=1}^{m_2} \frac{V_{i,j} - V_{e,k}}{h_2}, \quad (\text{A.69})$$

$$\left. \frac{\partial V}{\partial y} \right|_{(i,j+1/2)} = \sum_{k=1}^{m_3} \frac{V_{c,k} - V_{i,j}}{h_3}, \quad (\text{A.70})$$

$$\left. \frac{\partial V}{\partial y} \right|_{(i,j-1/2)} = \sum_{k=1}^{m_4} \frac{V_{i,j} - V_{b,k}}{h_4}, \quad (\text{A.71})$$

onde m_1 é o número de vizinhos à direita da célula centrada em (i, j) , m_2 é o número de vizinhos à esquerda, m_3 é o número de vizinhos acima, m_4 é o número de vizinhos abaixo e $V_{d,k}$ são os vizinhos da direita, $V_{e,k}$ são os vizinhos da esquerda, $V_{c,k}$ são os vizinhos de cima e $V_{b,k}$ são os vizinhos de baixo e

$$\begin{aligned} h_1 &= h_{i,j} \text{ se } \mathcal{L}_{i,j} > \mathcal{L}_{d,k} \text{ e } h_1 = h_{d,k} \text{ caso contrário,} \\ h_2 &= h_{i,j} \text{ se } \mathcal{L}_{i,j} > \mathcal{L}_{e,k} \text{ e } h_1 = h_{e,k} \text{ caso contrário,} \\ h_3 &= h_{i,j} \text{ se } \mathcal{L}_{i,j} > \mathcal{L}_{c,k} \text{ e } h_1 = h_{c,k} \text{ caso contrário,} \\ h_4 &= h_{i,j} \text{ se } \mathcal{L}_{i,j} > \mathcal{L}_{b,k} \text{ e } h_1 = h_{b,k} \text{ caso contrário,} \end{aligned} \quad (\text{A.72})$$

onde $\mathcal{L}$ é o nível de refinamento da célula na malha. Rearranjando e substituindo as discretizações na Eq. (2.39) e decompondo os operadores diferenciais temos:

$$C_m \frac{V_{i,j}^* - V_{i,j}^n}{\Delta t} = \frac{(L_1 J_{x_{i+1/2,j}}^* - L_2 J_{x_{i-1/2,j}}^* + L_3 J_{y_{i,j+1/2}}^* - L_4 J_{y_{i,j-1/2}}^*)}{\beta A_{i,j}} \quad (\text{A.73})$$

$$C_m \frac{V_{i,j}^{n+1} - V_{i,j}^*}{\Delta t} = -I_{ion}(V_{i,j}^*, \boldsymbol{\eta}^n) \quad (\text{A.74})$$

$$\frac{\partial \boldsymbol{\eta}^{n+1}}{\partial t} = f(\boldsymbol{\eta}^n, V^*, t) \quad (\text{A.75})$$

onde:

$$L_1 J_{x_{i+1/2,j}} = \sigma_{x_{i+1/2,j}} \sum_{k=1}^{m_1} \frac{V_{d,k} - V_{i,j}}{h_1} L_1 \quad (\text{A.76})$$

$$L_2 J_{x_{i-1/2,j}} = \sigma_{x_{i-1/2,j}} \sum_{k=1}^{m_2} \frac{V_{i,j} - V_{e,k}}{h_2} L_2 \quad (\text{A.77})$$

$$L_3 J_{y_{i,j+1/2}} = \sigma_{y_{i,j+1/2}} \sum_{k=1}^{m_3} \frac{V_{c,k} - V_{i,j}}{h_3} L_3 \quad (\text{A.78})$$

$$L_4 J_{y_{i,j-1/2}} = \sigma_{y_{i,j-1/2}} \sum_{k=1}^{m_4} \frac{V_{i,j} - V_{b,k}}{h_4} L_4. \quad (\text{A.79})$$

No caso de uma malha regular temos, $L_1 = h_1$, $L_2 = h_2$, $L_3 = h_3$, $L_4 = h_4$. Sendo assim podemos simplificar as equações acima, obtendo:

$$J_{x_{i+1/2,j}} = \sum_{k=1}^{m_1} \sigma_{x_{d',k}} (V_{d,k} - V_{i,j}) \quad (\text{A.80})$$

$$J_{x_{i-1/2,j}} = \sum_{k=1}^{m_2} \sigma_{x_{e',k}} (V_{i,j} - V_{e,k}) \quad (\text{A.81})$$

$$J_{y_{i,j+1/2}} = \sum_{k=1}^{m_3} \sigma_{y_{c',k}} (V_{c,k}^{n+1} - V_{i,j}) \quad (\text{A.82})$$

$$J_{y_{i,j-1/2}} = \sum_{k=1}^{m_4} \sigma_{y_{b',k}} (V_{i,j} - V_{b,k}) \quad (\text{A.83})$$

onde $\sigma_{x_{d',k}}$, $\sigma_{x_{e',k}}$, $\sigma_{y_{c',k}}$ e $\sigma_{y_{b',k}}$ são os valores das condutividades nas bordas do elemento e são calculados usando média harmônica, como segue:

$$\sigma_{x_{d',k}} = \frac{2\sigma_{x_{i,j}}\sigma_{x_{d,k}}}{\sigma_{x_{d,k}} + \sigma_{x_{i,j}}} \quad (\text{A.84})$$

$$\sigma_{x_{e',k}} = \frac{2\sigma_{x_{i,j}}\sigma_{x_{e,k}}}{\sigma_{x_{e,k}} + \sigma_{x_{i,j}}} \quad (\text{A.85})$$

sendo $\sigma_{x_{d',k}}$ e $\sigma_{x_{e',k}}$ as condutividades dos elementos k a direita e a esquerda do elemento em (i, j) respectivamente. $\sigma_{y_{c',k}}$ e $\sigma_{y_{b',k}}$ são calculado de maneira análoga.