RENATO SEIJI TAVARES

Reconstrução de Imagens por Tomografia por Impedância Elétrica
Utilizando Recozimento Simulado Massivamente Paralelizado

São Paulo
2016

RENATO SEIJI TAVARES

Reconstrução de Imagens por Tomografia por Impedância Elétrica
Utilizando Recozimento Simulado Massivamente Paralelizado

Tese apresentada à Escola Politécnica
da Universidade de São Paulo para a
obtenção do título de Doutor em Ciências

Área de Concentração:
Engenharia Mecânica

Orientador: Prof. Dr. Marcos de Sales
Guerra Tsuzuki
Co-Orientador: Prof. Dr. Thiago de Castro
Martins

São Paulo
2016

Este exemplar foi revisado e corrigido em relação à versão original, sob responsabilidade única do autor e com a anuência de seu orientador.

São Paulo, _____ de _____ de _____

Assinatura do autor: _____

Assinatura do orientador: _____

Catálogo-na-publicação

Tavares, Renato Seiji
Reconstrução de Imagens por Tomografia por Impedância Elétrica
Utilizando Recozimento Simulado Massivamente Paralelizado / R. S. Tavares
- versão corr. -- São Paulo, 2016.
138 p.

Tese (Doutorado) - Escola Politécnica da Universidade de São Paulo.
Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos.

1.TOMOGRAFIA 2.PROCESSAMENTO DE IMAGENS 3.OTIMIZAÇÃO
ESTOCÁSTICA 4.PROBLEMAS INVERSOS I.Universidade de São Paulo.
Escola Politécnica. Departamento de Engenharia Mecatrônica e de Sistemas
Mecânicos II.t.

Este trabalho é dedicado a minha família, namorada, amigos e professores, que me deram grande apoio e incentivo durante a jornada acadêmica.

Ao Prof. Dr. Goki Tsuzuki e a sua esposa, Araci Tsuzuki, por todo incentivo, motivação e confiança.

A todos que sonham se tornar cientistas e se dedicam a seguir o caminho da ciência e da educação apesar de todas as dificuldades.

E a todos aqueles que buscam sempre se aprimorar e enfrentam todos os desafios.

Agradecimentos

Agradeço ao Prof. Dr. Marcos de Sales Guerra Tsuzuki pelas orientações durante a realização do doutorado, por ter me guiado pelos caminhos da pesquisa, por seus ensinamentos e pelas oportunidades oferecidas.

Agradeço ao Prof. Dr. Thiago de Castro Martins pela orientação e pelo grande auxílio no desenvolvimento da pesquisa.

Agradeço ao Dr. André Kubagawa Sato, ao Edson Ueda e a todos os meus demais colegas de laboratório por todo o auxílio que recebi e por todos os momentos que passamos juntos.

Agradeço a minha família e amigos que me apoiaram e incentivaram ao longo de todos estes anos.

Agradeço a FAPESP, pela bolsa de estudos concedida para a realização desta pesquisa.

Por fim, agradeço a minha namorada, Amanda Ribeiro, por toda a dedicação e, principalmente, pela enorme paciência durante todos estes anos.

Resumo

A tomografia por impedância elétrica é uma modalidade de imageamento médico recente, com diversas vantagens sobre as demais modalidades já consolidadas. O recozimento simulado é um algoritmo que apresenta qualidade de solução, mesmo com a utilização de uma regularização simples e sem informação a priori. Entretanto, existe a necessidade de reduzir o tempo de processamento. Este trabalho avança nessa direção, com a apresentação de um método de reconstrução que utiliza o recozimento simulado e paralelização massiva em GPU. A paralelização das operações matriciais em GPU é explicada, com uma estratégia de agendamento de threads que permite a paralelização efetiva de algoritmos, até então, considerados não paralelizáveis. Técnicas para sua aceleração são discutidas, como a heurística de fora para dentro. É proposta uma nova representação de matrizes esparsas voltada para as características da arquitetura CUDA, visando um melhor acesso à memória global do dispositivo e melhor utilização das threads. Esta nova representação de matriz mostrou-se vantajosa em relação aos formatos mais utilizados. Em seguida, a paralelização massiva do problema inverso da TIE, utilizando recozimento simulado, é estudada, com uma proposta de abordagem híbrida com paralelização tanto em CPU quanto GPU. Os resultados obtidos para a paralelização do problema inverso são superiores aos do problema direto. A GPU satura em aproximadamente 7.000 nós, a partir do qual o ganho em desempenho é de aproximadamente 5 vezes. A utilização de GPUs é viável para a reconstrução de imagens de tomografia por impedância elétrica.

Palavras-chave: Tomografia por Impedância Elétrica. GPU. CUDA. Algoritmos Paralelos. Recozimento Simulado.

Abstract

Electrical impedance tomography is a new medical imaging modality with remarkable advantages over other established modalities. Simulated annealing is an algorithm that renders quality solutions despite the use of simple regularization methods and the absence of a priori information. However, it remains the need to reduce its processing time. This work takes a step in this direction, presenting a method for the reconstruction of EIT images using simulated annealing and GPU parallelization. The parallelization of matrix operations in GPU is explained, with a thread scheduling strategy that allows the effective parallelization of not-yet effectively parallelized algorithms. There are strategies for improving its performance, such as the presented outside-in heuristic. It is proposed a new sparse matrix representation focused on the CUDA architecture characteristics, with improved global memory access patterns and thread efficiency. This new matrix representation showed several advantages over the most common formats. The massive parallelization of the TIE's inverse problem using simulated annealing is studied, with a proposed hybrid approach that uses parallelization in both CPU and GPU. Results showed that the performance gain for the inverse problem is higher than the one obtained for the forward problem. The GPU device saturates with meshes of size of approximately 7,000 nodes, with a performance gain around 5 times faster than serial implementations. GPU parallelization may be used for the reconstruction of electrical impedance tomography images.

Keywords: Electrical impedance tomography. GPU. CUDA. Parallelized Algorithms. Simulated Annealing.

Lista de ilustrações

Figura 1 – Ilustração de uma imagem obtida através da TIE. (a) Foto do aparato experimental, com 3 pepinos em uma cuba preenchida com solução salina. (b) Imagem reconstruída.	22
Figura 2 – Ilustração de 4 dos 32 padrões de corrente. Os triângulos, na borda, representam os 32 eletrodos. (a) Primeiro padrão de corrente, com injeção de corrente nos eletrodos 1 e 5. (b) Segundo padrão de corrente, com injeção de corrente nos eletrodos 2 e 6. (c) Terceiro padrão de corrente, com injeção de corrente nos eletrodos 3 e 7. (d) Trigésimo segundo, e último, padrão de corrente, com injeção de corrente nos eletrodos 32 e 4.	24
Figura 3 – (a) Uma malha com 30.216 nós, com alta densidade de nós na região circular interna e eletrodos simples. (b) Detalhe da diferença no refinamento das regiões interna e anel externo, para a malha mostrada em (a).	30
Figura 4 – (a) Uma malha com 29.074 nós, com eletrodos com alta densidade de nós e região circular interna com baixa densidade. (b) Detalhe da diferença no refinamento das regiões interna e anel externo, para a malha mostrada em (a).	31
Figura 5 – (a) Uma malha com 29.517 nós, com densidade média de nós nas regiões circular interna e do anel externo (e eletrodos). (b) Detalhe da diferença no refinamento das regiões interna e anel externo, para a malha mostrada em (a).	32
Figura 6 – Elemento isoparamétrico triangular, com nós numerados no sentido anti-horário.	34
Figura 7 – Modelo de eletrodo, com 3 nós na base.	34
Figura 8 – Modelo de eletrodo simplificado, com base de 3 nós.	35
Figura 9 – Modelo de eletrodo genérico.	36
Figura 10 – Uma subdivisão do eletrodo.	36
Figura 11 – Modelo de eletrodo genérico simplificado.	37
Figura 12 – Espaço de malhas exploradas, considerando o número de nós no anel externo e nas regiões circulares internas. N _{NER} = Número de nós no anel externo, N _{NIC} = Número de nós na região circular interna. A cor indica o número total de nós.	39
Figura 13 – Distribuição do potencial elétrico para a malha de referência.	39
Figura 14 – Potenciais elétricos nos eletrodos, para a malha de referência.	40

Figura 15 – Gráfico mostrando a relação entre o número total de nós da malha (horizontal) e a razão entre os números de nós nas regiões do anel externo e na região circular interna (vertical). Cor indica o módulo de ν (mV).	40
Figura 16 – Gráfico mostrando os erros nos potenciais (em cor) nos eletrodos ativos – eletrodos em que são aplicadas correntes – associados ao número de nós no anel externo (vertical) e na região circular interna (horizontal).	41
Figura 17 – Gráfico mostrando os erros nos potenciais (em cor) nos eletrodos passivos – eletrodos em que não são aplicadas correntes – associados ao número de nós no anel externo (vertical) e na região circular interna (horizontal).	42
Figura 18 – Ilustração da arquitetura de memória CUDA.	49
Figura 19 – Diferenças entre acessos alinhado e desalinhado à memória global na arquitetura CUDA. (a) Acesso alinhado precisa de quatro (sem cache) ou apenas uma (acesso no cache) operação de leitura de memória para ler 32 valores. (b) No caso de um acesso desalinhado, são necessárias cinco (sem cache) ou dois (acesso no cache) ciclos de leitura.	51
Figura 20 – Os elementos não nulos de cada linha são pintados da mesma cor. Cada linha é armazenada como um vetor esparsos, e todas as linhas (isto é, todos os vetores esparsos) são armazenados de forma contígua em v e ci . O vetor p indica onde cada vetor esparsos começa em v e ci	52
Figura 21 – Comparação do acesso à memória nos formatos ELLPACK-R and pJDS - o formato pJDS adiciona uma quantidade menor de “enchimento”. (a) Matriz exemplo, de tamanho 12×12 - a cor vermelha indica elementos não-nulos. (b) Matriz no formato ELLPACK-R: os elementos são compactados à esquerda, e zeros de enchimento são adicionados para que as linhas do mesmo warp possuam o mesmo comprimento (estes estão indicados na cor vermelho claro). (c) Matriz no formato pJDS: similar ao formato ELLPACK-R, exceto que as linhas são reordenadas de acordo com o tamanho antes da realização do enchimento com zeros. Warp hipotético de tamanho 2 threads.	57
Figura 22 – Modelo MEF com 5 nós numerados sequencialmente e sua matriz de rigidez. As cores indicam grau de dependência.	58
Figura 23 – Matriz triangular inferior obtida após a aplicação do pré-condicionador à matriz de rigidez do modelo MEF simples da Fig. 22.	58
Figura 24 – O mesmo MEF com 5 nós mostrado na Fig. 22, renumerado, e sua nova matriz de rigidez. Novamente, as cores indicam grau de dependência, com uma redução do número de cores necessárias.	59
Figura 25 – Matriz triangular inferior obtida após a aplicação do pré-condicionador à matriz de rigidez do modelo MEF ordenado por cor da Fig. 24.	59

Figura 26 – Procedimento para criar o formato pJDS colorido. A estrela indica o elemento da diagonal. (a) Matriz da Fig. 21 colorida. (b) Matriz colorida com elementos não-nulos compactados à esquerda. (c) Matriz com as linhas reordenadas de acordo com a cor (primariamente) e tamanho (secundariamente). (d) Preenchimento com zeros – adicionados para que todas as linhas acessadas por threads no mesmo warp possuam o mesmo tamanho, garantindo que todas as threads leiam blocos inteiros naturalmente alinhados, mostrados na cor mais clara, com setas indicando o crescimento da linha. Além disso, linhas também são adicionadas, para que todos os grupos de cor possuam tamanho múltiplo do tamanho do warp (warp hipotético de 2 threads). Por fim, os elementos da diagonal são artificialmente colocados na primeira posição da linha, deslocando o resto da linha.	62
Figura 27 – Multiplicação ponto a ponto do produto interno. Cada thread realiza uma única multiplicação, e armazena o produto em um vetor de resultado parcial.	64
Figura 28 – Redução do vetor de resultados parciais do produto interno. O número de threads ativas cai sempre pela metade. Neste exemplo, considerou-se a redução de um vetor de tamanho 16.	65
Figura 29 – Iterações por coluna na multiplicação matriz esparsa $\times$ vetor. As colunas com cor escura indicam as leituras de memória – realizadas em blocos contíguos naturalmente alinhados. As threads ativas em cada iteração estão apresentadas à esquerda. São necessárias 7 iterações para multiplicar esta matriz de lado 14 por um vetor.	67
Figura 30 – Iterações por grupo de cor no solver triangular inferior e superior. Os blocos com cor escura indicam as leituras de memória – realizadas em blocos contíguos naturalmente alinhados. Cada linha é processada por uma thread. São necessárias 6 chamadas do kernel de solver triangular.	70
Figura 31 – Decomposição incompleta de Cholesky (a) progressiva e serial; (b) em paralelo, devido à coloração.	74
Figura 32 – Linha do tempo de execução de um kernel vazio.	75
Figura 33 – Linha do tempo de execução de um kernel vazio – espaçamento entre blocos de execução indicam GPU ociosa.	76
Figura 34 – Linha do tempo de execução para o produto matriz esparsa $\times$ vetor. Espaçamento entre blocos de execução indicam GPU ociosa, desta vez o gargalo está na CPU.	77
Figura 35 – Comparação do tempo em relação ao número de nós para a multiplicação matriz esparsa $\times$ vetor, Eigen vs. CUDA com o formato de matriz proposto. Tempo em relação ao número de nós.	78

Figura 36 – Linha do tempo de execução para o solver triangular – inferior e superior. Espaço entre blocos de execução indicam GPU ociosa.	79
Figura 37 – Comparação do tempo em relação ao número de nós para o solver triangular completo – inferior e superior, Eigen vs. CUDA com o formato de matriz proposto.	79
Figura 38 – Diagrama de sequência do algoritmo dos métodos dos Gradientes Conjugados - inicialização.	80
Figura 39 – Diagrama de sequência de execução do solver triangular. O kernel é invocado uma vez para cada cor. A quantidade de threads pode ser igual ao número de elementos da cor. As setas significam threads. . . .	81
Figura 40 – Diagrama de sequência de execução do solver triangular ilustrando o problema da falta de sincronização entre as cores. O kernel é invocado uma única vez e a iteração por cor foi migrada para dentro do kernel. A quantidade de threads utilizadas é igual ao tamanho da matriz. As setas significam threads.	82
Figura 41 – Diagrama de sequência de execução do solver triangular, utilizando um único bloco. O kernel é invocado uma única vez e a iteração por cor foi migrada para dentro do kernel. A quantidade de threads utilizadas é igual ao tamanho da matriz. As setas significam threads.	83
Figura 42 – Diagrama de sequência do algoritmo dos Gradientes Conjugados - iteração.	84
Figura 43 – Diagrama de sequência reordenado do algoritmo dos Gradientes Conjugados - iteração.	85
Figura 44 – Linha do tempo de execução para o algoritmo de gradientes conjugados pré-condicionado utilizando consolidação em 3 kernels.	87
Figura 45 – Comparação do tempo em relação ao número de nós para o algoritmo dos Gradientes Conjugados pré-condicionado, Eigen vs. CUDA com o formato de matriz proposto. Média para 30 iterações.	87
Figura 46 – Comparação do ganho de velocidade no problema direto paralelizado em GPU proposto em relação à implementação serial usando Eigen. . .	88
Figura 47 – Distribuição de potencial elétrico – resultado do problema direto paralelizado para a malha de 2.590 nós e padrão de corrente “pula três”. . .	88
Figura 48 – Comparação do tempo em relação ao número de nós para a decomposição de Cholesky incompleta, Eigen vs. CUDA com o formato de matriz proposto.	89
Figura 49 – Gráfico da temperatura pela energia para o recozimento simulado proposto por Corana et al. (CORANA et al., 1987), onde a próxima solução candidata é selecionada segundo uma distribuição de probabilidades plana e o máximo valor possível diminui com a temperatura.	93

Figura 50 – Gráfico da temperatura pela energia para o recozimento simulado proposto por Ingber (INGBER, 1996), onde a próxima solução candidata é selecionada segundo uma distribuição de probabilidades de Cauchy e o máximo valor possível permanece inalterado com a temperatura.	94
Figura 51 – Exemplo de malha de elementos finitos, representando a hierarquia das informações. No contorno tem-se mais informações e estas condutividades devem ser estimadas primeiro.	97
Figura 52 – Representação esquemática do aparato experimental.	98
Figura 53 – Foto do aparato experimental.	98
Figura 54 – Malha utilizada na reconstrução de imagens a partir dos dados experimentais.	99
Figura 55 – Variação dos pesos para sorteio de cada camada, conforme a temperatura.	100
Figura 56 – Variação das probabilidades de sorteio de cada camada, conforme a temperatura.	101
Figura 57 – Variação do sorteio de cada camada conforme a temperatura.	101
Figura 58 – Imagem reconstruída a partir dos dados experimentais, utilizando a heurística de fora para dentro. Escala apresenta valores de potencial elétrico, em mV.	102
Figura 59 – Iterações de Gradientes Conjugados por temperatura, para a reconstrução a partir de dados experimentais - com e sem a heurística de fora para dentro.	102
Figura 60 – Taxa de aceite de candidatos por temperatura - com e sem a heurística de fora para dentro.	103
Figura 61 – Diagrama de sequência para a implementação serial do recozimento interno (ver Algoritmo 17).	105
Figura 62 – Ilustração do laço interno proposto para a paralelização do recozimento interno (ver Algoritmo 17).	107
Figura 63 – Experimento com três pepinos dispostos em padrão triangular.	108
Figura 64 – Reconstrução de imagem de TIE com três pepinos, utilizando a implementação serial. Escala apresenta valores de potencial elétrico, em mV.	108
Figura 65 – Iterações dos Gradientes Conjugados com avaliação incompleta da função objetivo por temperatura.	109
Figura 66 – Decaimento da função objetivo pela temperatura.	109
Figura 67 – Taxa de aceite de soluções pela temperatura.	110
Figura 68 – Reconstrução de imagem de TIE com três pepinos, utilizando a implementação massivamente paralelizada proposta. Potencial elétrico em mV.	110

Figura 69 – Linha do tempo de chamadas da API CUDA, ilustrando a utilização de uma thread para cada padrão de corrente para a invocação de kernels.	111
Figura 70 – Linha do tempo de execução dos kernels do problema direto, ilustrando a utilização de um stream para cada padrão de corrente.	112
Figura 71 – Ganho de desempenho no problema inverso utilizando a abordagem proposta, em relação ao número de nós.	112
Figura 72 – Domínio de simulação.	135
Figura 73 – Comportamento da convergência do problema do tabuleiro de xadrez. .	136
Figura 74 – Energia por temperatura determinada para o problema do tabuleiro de xadrez (as barras de erro representam o desvio padrão da energia para a temperatura).	136
Figura 75 – Iterações de Gradientes Conjugados (vertical) x temperatura (horizontal) para o problema do tabuleiro de xadrez.	138

Lista de tabelas

Tabela 1 – Tempo total médio de execução (em ms) para o produto interno - 1.000 repetições.	76
Tabela 2 – Tempo total médio de execução (em ms) para a multiplicação matriz esparsa $\times$ vetor – 1.000 repetições.	77
Tabela 3 – Tempo total médio de execução (em ms) para a multiplicação matriz esparsa $\times$ vetor – 1.000 repetições.	78
Tabela 4 – Tempo total médio de execução (em ms) para o solver triangular - 1.000 repetições.	78
Tabela 5 – Comparação dos algoritmos dos Gradientes Conjugados Pré-condicionados - tempo médio (em ms) para 30 iterações	86
Tabela 6 – Comparação dos algoritmos de decomposição de Cholesky incompleta – tempo em ms	87
Tabela 7 – Pré-processamento - tempo médio para 100 iterações	89
Tabela 8 – Comparação do tempo de execução da avaliação de função objetivo, considerando 30 iterações dos Gradientes Conjugados – tempo médio (em ms)	111
Tabela 9 – Comparação do tempo de processamento para a regularização – tempo médio (em μs) efetivo para a CPU	113
Tabela 10 – Comparação do tempo de processamento e transferência de memória do pré-cálculo da próxima solução – tempo médio (em ms)	113

Lista de abreviaturas e siglas

BCSR	Block Compressed Sparse Row
COO	Coordinates Formt (compressão de matriz esparsa por coordenadas, em inglês)
CpJDS	Colored Padded Jagged Diagonal Storage
CPU	Central Processing Unit (unidade de processamento central, em inglês)
CSC	Compressed Sparse Column (compressão de matriz esparsa por coluna, em inglês)
CSR	Compressed Sparse Row (compressão de matriz esparsa por linha, em inglês)
CUDA	Compute Unified Device Archictecture, um framework para programação em GPU
GPGPU	General-Purpose computation on GPU (computação de uso geral em GPU, em inglês)
GPU	Graphics Processing Unit (unidade de processamento gráfico, em inglês)
JDS	Jagged Diagonal Storage
MEF	Método dos Elementos Finitos
MRI	Magnetic Ressonance Imaging (imageamento por ressonância magnética, em inglês)
pJDS	Padded Jagged Diagonal Storage
TIE	Tomografia por Impedância Elétrica

Lista de símbolos

B	Densidade do fluxo magnético
C	Matriz com todos os padrões de corrente, linearmente independentes
E	Campo elétrico
ϵ	Permissividade
γ	Distribuição de condutâncias no domínio
I	Intensidade da corrente elétrica injetada no domínio
J	Corrente elétrica injetada no domínio, como um vetor
K	Matriz de rigidez
k	Elemento da matriz de rigidez
Φ	Matriz com todos os potenciais elétricos nodais, para todos os padrões de corrente
ϕ	Distribuição de potencial elétrico
ψ	Função de interpolação
ρ	Densidade de carga elétrica
Ω	Domínio

Sumário

1	INTRODUÇÃO	17
	Introdução	17
1.1	Motivação	17
1.2	Sobre esta Pesquisa	19
1.3	Organização do Texto	20
2	TOMOGRAFIA POR IMPEDÂNCIA ELÉTRICA	21
2.1	Introdução	21
2.2	Conceitos Básicos	23
2.3	Imagem Absoluta e Imagem Relativa	26
2.4	A TIE e seus Algoritmos de Reconstrução	27
2.5	O Problema Direto da TIE	28
2.6	Malhas para o Modelo de Elementos Finitos (MEF)	29
2.6.1	Criação da malha	29
2.6.2	Determinação da Matriz de Condutividade	33
2.6.3	Modelo de Eletrodo	34
2.6.4	Modelo de Eletrodo com Número de Elementos Variável	36
2.7	Estudo do Erro das Malhas	37
2.7.1	Análise do Erro	38
2.8	O Algoritmo de Gradientes Conjugados	42
2.8.1	Convergência	43
2.8.2	Pré-Condicionamento	43
2.8.3	Decomposição de Cholesky	45
2.8.3.1	Decomposição de Cholesky Incompleta	45
2.9	Conclusões	46
3	PARALELIZAÇÃO DO PROBLEMA DIRETO DA TIE EM GPU	48
3.1	Estruturas de Dados para Matrizes Esparsas e Multiplicação matriz esparsa $\times$ vetor	50
3.2	Paralelização de Solvers Triangulares através de Coloração de Grafo	56
3.3	O formato pJDS Colorido para Armazenamento da Matriz	60
3.4	Operações Paralelizadas com Matriz e Vetor	63
3.4.1	Produto Interno de 2 Vetores	63
3.4.2	Multiplicação Matriz esparsa $\times$ Vetor	67
3.4.3	Solver Triangular	68

3.5	Gradientes Conjugados Paralelizado	71
3.6	Decomposição de Cholesky Incompleta	73
3.7	Resultados	73
3.7.1	Medindo Tempo de Execução	74
3.7.2	Avaliando os Kernels Individualmente	76
3.8	Consolidação de Kernels	80
3.9	Resultados para Kernels Consolidados	86
3.10	Conclusão	89
4	RECONSTRUÇÃO DE IMAGENS DE TIE ATRAVÉS DO PROBLEMA INVERSO	91
4.1	Recozimento Simulado	91
4.2	Aplicando o Recozimento Simulado à TIE	95
4.2.1	Regularização através da Norma do Gradiente	95
4.3	Heurística de Fora para Dentro	96
4.3.1	Utilizando dados experimentais	97
4.3.2	Forçando a caminhada de fora para dentro	98
5	PARALELIZAÇÃO DO PROBLEMA INVERSO DA TIE USANDO GPU	104
5.1	Recozimento Simulado Paralelizado	104
5.2	Resultados	106
5.3	Conclusão	114
6	CONCLUSÕES E TRABALHOS FUTUROS	115
	REFERÊNCIAS	118
	APÊNDICES	126
	APÊNDICE A – FORMULAÇÃO MATEMÁTICA DA TIE	127
A.1	Formulação Variacional	129
A.2	Funções de Interpolação	129
A.3	Montagem do Sistema de Equações Lineares do Modelo	130
A.4	Cálculo Analítico das Constantes do Sistema	131
	APÊNDICE B – AVALIAÇÃO INCOMPLETA DA FUNÇÃO OBJETIVO	133
B.1	Resultados Obtidos pela Implementação do Método	135
B.2	Problema do Tabuleiro de Xadrez	136

1 Introdução

Neste capítulo, apresenta-se a motivação da pesquisa, detalhando sobre as formas da Tomografia por impedância elétrica (TIE) e suas aplicações.

1.1 Motivação

A Tomografia por Impedância (TIE) é uma modalidade de imageamento que estima a distribuição de condutividade elétrica em um corpo quando um padrão de corrente com baixa amplitude é aplicado à superfície do corpo e o potencial em alguns pontos determinados da superfície são medidos por meio de eletrodos ou, alternativamente, quando um potencial é aplicado e o fluxo de corrente através da superfície é medido (TRIGO; LIMA; AMATO, 2004).

O problema fundamental na reconstrução de imagens pela TIE é, no caso geral, que a corrente elétrica não pode ser forçada a fluir linearmente (ou mesmo ao longo de uma trajetória conhecida) em um condutor não homogêneo; o fluxo de corrente é determinado pela distribuição de condutividade elétrica pelo corpo. O modelo do corpo é baseado nas equações diferenciais parciais elípticas obtidas a partir das equações de Maxwell e a distribuição de condutividade elétrica representa a solução de um problema inverso não linear mal posto, significando que grandes modificações na condutividade do interior do corpo podem resultar em apenas pequenas modificações de potencial ou corrente na superfície. Os principais problemas com a TIE são: a sua sensibilidade ao posicionamento do eletrodo, a baixa resolução da sua imagem, e a sua sensibilidade à geometria do contorno.

As duas principais formas de TIE são, como em outras técnicas de imageamento médico como ressonância magnética (RM) e tomografia computadorizada (TC), imageamento estático e dinâmico, sendo que ambas as formas podem produzir imagens absolutas e relativas. Imagens relativas representam as mudanças de condutividade de uma região entre dois instantes no tempo. O imageamento fisiológico funcional de um corpo é feito utilizando esta técnica. Neste trabalho, foca-se na reconstrução de imagens estáticas absolutas que requerem algoritmos numéricos mais avançados (BARBER; BROWN, 1984).

A resolução espacial da TIE não é comparável a outras técnicas de imageamento como RM ou TC. Entretanto, a TIE apresenta algumas vantagens sobre estas técnicas, como não produzir efeitos colaterais ao paciente, a de apresentar baixo custo e a de ser portátil. A TIE também possui características de rápida resposta temporal, que permite um melhor monitoramento de mudanças cíclicas em tecidos vivos do que outras técnicas

de imageamento. Muitos pesquisadores estão realizando esforços contínuos para conseguir algoritmos que são, ao mesmo tempo, velozes e capazes de fornecer imagens com ótima resolução espacial.

A TIE possui uma grande variedade de aplicações médicas, tais como detecção de acidente vascular cerebral agudo (CLAY; FRREE, 2002), câncer de mama (KAO et al., 2006), monitoramento das atividades cardíacas (EYÜBOGLU; BROWN; BARBER, 1989) e monitoramento da aeração pulmonar imposta por ventilação mecânica em pacientes críticos (HUA et al., 1993; TRIGO; LIMA; AMATO, 2004). Esta última aplicação é o maior interesse deste projeto de pesquisa, onde a TIE pode ser utilizada para implementar uma estratégia de proteção do pulmão que pode resultar em melhora da função pulmonar e sobrevida do paciente (AMATO et al., 1998).

O primeiro algoritmo para gerar imagens pulmonares utilizando a TIE faz uso de uma aproximação linear para resolver o problema inverso da TIE chamado de “backprojection”. Esta aproximação impõe algumas restrições como geometria circular do domínio e máxima variação de impedância que pode ser estimada com precisão. Vários algoritmos iterativos foram propostos, aumentando a precisão dos resultados ao considerar a TIE como um problema não linear. Ambos os algoritmos não são excludentes, e a saída de algoritmos lineares pode ser utilizada como entrada para algoritmos iterativos e vice-versa. Enquanto um prioriza a velocidade, o segundo prioriza a precisão de estimativas quando se dispõe de um maior tempo de análise.

O recozimento simulado é uma das técnicas utilizadas para obter imagens absolutas que tem produzido ótimos resultados na literatura. O recozimento simulado é um algoritmo de heurística probabilística que possui duas fases principais: exploração e refinamento. O recozimento simulado utilizado nesta pesquisa, desenvolvido pelo grupo do Laboratório de Geometria Computacional, permite que alguns parâmetros estejam na fase de refinamento enquanto outros estão na fase exploratória. Este grau de liberdade é necessário na solução de alguns problemas em que existam parâmetros que devem ser definidos antes que outros.

No caso particular da TIE, o problema, como mencionado, consiste em encontrar as condutividades elétricas internas a um corpo a partir de medidas realizadas em seu contorno. Um conjunto de correntes elétricas é aplicado à superfície do corpo e as voltagens resultantes são medidas sobre a sua superfície. Através do recozimento simulado, é possível reconstruir o mapa de condutividades no interior do corpo. A característica do algoritmo de permitir regiões em diferentes fases pode ser extrapolada para forçar um sentido na reconstrução. Isto é, o algoritmo prioriza a condutividade próxima ao contorno do corpo e reduz a prioridade para a condutividade em seu interior. Assim, a condutividade próxima ao contorno do corpo é detalhada primeiro, e durante este detalhamento a condutividade do interior ainda está na fase exploratória.

1.2 Sobre esta Pesquisa

Para a utilização do recozimento simulado para solucionar o problema inverso da TIE, necessita-se acelerar o processamento computacional. Esta tese foca na redução deste tempo computacional. Foram estudadas duas estratégias: criação de uma heurística de convergência nomeada como heurística de fora para dentro (apresentada na seção 4.3), e a utilização de placas gráficas (GPU) com o objetivo de paralelizar massivamente o processamento.

A redução efetiva de tempo computacional ocorreu com a utilização de placas gráficas, e não com a heurística de fora para dentro. Foram feitos dois estudos principais com a utilização das placas gráficas: solução do problema direto (onde a condutividade do domínio e as correntes aplicadas são conhecidas, e determina-se os potenciais), e solução do problema inverso (onde as correntes e os potenciais são conhecidos e determina-se a distribuição de condutividade).

Para que a GPU fosse utilizada efetivamente, evitando acessos desalinhados à memória e divergência de threads, foi proposta uma nova representação de matriz esparsa, voltada para a repetição por colunas e com a utilização de cores para agrupar dependências. Todos os kernels necessários para a implementação dos Gradientes Conjugados foram implementados. Apesar de todos estes avanços, foi identificado que a GPU permanecia em espera a maior parte do tempo, mesmo todos os kernels sendo enviados um atrás do outro. Optou-se por outra solução onde todos os kernels foram aglutinados, e kernels que precisavam de um sincronismo, como o produto interno entre vetores, foram separados em duas etapas. Desta maneira, conseguiu-se um ganho efetivo em tempo computacional.

Para a solução do problema inverso, considerando que o dispositivo de TIE possui 32 eletrodos, é necessário resolver simultaneamente 32 sistemas lineares. Estes problemas, porém, são independentes entre si e podem ser enviados separadamente para processamento em GPU. A regularização utilizada, a norma do gradiente da solução, é extremamente paralelizável e também pode ser resolvido em paralelo aos 32 problemas diretos. Além disso, o cálculo da próxima solução, com a necessidade de alteração da matriz de rigidez e o consequente cálculo do pré-condicionador, são operações sequenciais que podem ser evitados serem feitos em GPU – principalmente com o cálculo do pré-condicionador sendo tão ineficiente.

Assim, é apresentada uma abordagem híbrida, com o uso de paralelização massiva em GPU e a preparação de dados em CPU, obtendo-se um ganho adicional em desempenho em relação ao observado no problema direto.

1.3 Organização do Texto

Este texto está dividido da seguinte maneira.

O capítulo 2 apresenta a tomografia por impedância elétrica, seu problema direto da TIE, e o estudo de sua modelagem por elementos finitos.

O capítulo 3 apresenta a arquitetura CUDA, um novo formato para representar matrizes esparsas proposto levando-se em consideração as características da plataforma, e a paralelização do problema direto da TIE.

O capítulo 4 apresenta o problema inverso da TIE, o recozimento simulado, e sua aplicação na reconstrução de imagens de TIE. Também é apresentado um novo método para acelerar a convergência, a heurística de fora para dentro.

O capítulo 5 apresenta a paralelização do problema inverso utilizando as técnicas apresentadas no capítulo 3, com os requisitos adicionais para a reconstrução de imagens da TIE.

Por fim, no capítulo 6, são apresentadas as conclusões, considerações finais e possibilidades de trabalhos futuros.

2 Tomografia por Impedância Elétrica

Neste capítulo, os conceitos básicos da TIE serão apresentados, bem como as duas principais possibilidades de imageamento: imagem absoluta e imagem de diferenças. A formulação matemática será apresentada, sem considerar a presença dos efeitos capacitivos. Também será apresentada uma breve revisão sobre os algoritmos de reconstrução.

2.1 Introdução

Durante a segunda metade do século XX, novas e melhores técnicas foram desenvolvidas para se observar o interior do corpo humano. Desde então, as técnicas de tomografia vêm se consolidando como as mais importantes para a obtenção de imagens médicas (BATES; GARDENM; PETERS, 1983). As técnicas de tomografia são capazes de obter imagens da seção de um corpo sólido mediante a informação obtida de uma série de medidas em várias direções no plano da seção. O primeiro tomógrafo computadorizado surgiu na década de 70 e baseava-se na técnica de radiografia, descoberta por Wilhelm Conrad Röntgen (1895). As imagens são obtidas expondo a região do corpo que se quer analisar a uma radiação ionizante (raio-X) que atravessa o corpo e atinge uma placa fotográfica, que revelada mostra as sombras das regiões do corpo que atenuam a incidência dos raios-X (JAUNCEY, 1945). Na TC a imagem é obtida por meio de um algoritmo de reconstrução de imagem que utiliza a informação de radiografias de diferentes ângulos em torno da seção do corpo a ser tomografado (HERMAN, 1980). A RM é uma outra técnica de tomografia, ela utiliza os princípios da RM nuclear, em que o paciente é submetido a ondas de rádio e um campo magnético intenso. Esta técnica se baseia no princípio de que diferentes átomos do corpo humano absorvem diferentes frequências de ondas de rádio quando submetidas a um campo magnético. Um algoritmo de reconstrução é usado para obter as imagens a partir da informação de intensidade de absorção das ondas de rádio pelo corpo (HINSHAW; LENT, 1983).

Uma técnica bastante explorada desde o início da década de 80 e que vem se tornando uma alternativa em potencial às técnicas tradicionais é a TIE, que permite obter imagens de uma seção do corpo mediante valores de voltagem medidos em eletrodos posicionados ao redor do corpo (BARBER; BROWN, 1984; HENDERSON; WEBSTER, 1978; PRICE, 1979). A resolução das imagens geradas pela TIE é baixa, se comparada às outras técnicas de tomografia. Apesar disto, a TIE apresenta vantagens que a tornam viável para a aplicação clínica. Por exemplo, esta técnica não expõe a região do corpo que se quer observar a qualquer tipo de radiação, tem baixo custo em relação às outras técnicas de tomografia e, além disso, um tomógrafo de TIE é portátil e pode ser utilizado junto

Figura 1 – Ilustração de uma imagem obtida através da TIE. (a) Foto do aparato experimental, com 3 pepinos em uma cuba preenchida com solução salina. (b) Imagem reconstruída.

ao leito do paciente (BROWN, 2001; CHENEY et al., 1990). A TIE pode ser empregada tanto em aplicações médicas como industriais (BROWN, 2001).

Na tomografia, os algoritmos de reconstrução podem ser classificados em duas classes, segundo a maneira como as imagens de tomografia são determinadas: o problema direto e o problema inverso (CENSOR, 1983; LEWITT, 1983). No problema direto, dadas as fontes de fenômenos físicos e a distribuição de propriedades materiais da seção do corpo (cuja imagem é desejada), calcula-se o sinal da resposta correspondente aos estímulos. Por exemplo, na TIE o problema direto determina a distribuição de potenciais elétricos produzido, dado um carregamento elétrico aplicado e as propriedades de condutividade do material da seção do corpo (YORKEY; WEBSTER; TOMPKINS, 1987). Já no problema inverso, dado o sinal de resposta e as fontes de fenômenos físicos, reconstrói-se a distribuição de propriedades materiais da seção do corpo. Em técnicas de tomografia que utilizam radiação, é possível obter formulações inversas bem condicionadas (LEWITT, 1983).

Na TIE, a solução do problema inverso corresponde a se obter uma estimativa da condutividade (ou resistividade) da seção do corpo, a partir dos potenciais medidos no contorno dessa seção. Fig. 1 apresenta uma imagem de TIE reconstruída. Sabe-se que o problema inverso da TIE é mal-posto (GISSER; ISAACSON; NEWELL, 1990; SYLVESTER; UHLMANN, 1987) e essa característica faz com que a TIE seja bastante dependente do algoritmo de reconstrução utilizado. No problema inverso, a obtenção da solução exata é substituída por um outro problema mais bem condicionado - com o uso de regularização (ADLER; GUARDO, 1996; SCHMITT et al., 2002; TIKHONOV, 1991; WEST et al., 2001). Assim, vários algoritmos vêm sendo testados por vários pesquisadores nacionais e internacionais com o objetivo de identificar a melhor maneira, para cada caso de uso e condições, para a solução do problema inverso da TIE (BORCEA, 2002).

2.2 Conceitos Básicos

A TIE estima a distribuição de condutividades no interior de um corpo mediante informação de potenciais elétricos medidos na parte externa do corpo. Uma revisão sobre a teoria desta técnica de tomografia pode ser encontrada em (BORCEA, 2002). Uma cinta de eletrodos é colocada na superfície do corpo, em torno da seção transversal que se deseja observar. Aplica-se corrente elétrica a um par de eletrodos e os potenciais elétricos que surgem nos demais eletrodos são medidos e registrados por meio de um sistema de aquisição de dados (BARBER; BROWN, 1984).

A corrente elétrica é aplicada nos eletrodos seguindo certo padrão de excitação, chamado padrão de corrente, que pode ser adjacente, diametral ou algum outro. No padrão adjacente, a corrente elétrica é aplicada em dois eletrodos adjacentes (BROWN; SEAGAR, 1987; BARBER, 1989; ISAACSON; CHENEY; NEWELL, 1992). Já, no padrão diametral a corrente elétrica é aplicada em dois eletrodos diametralmente opostos (HUA; WEBSTER; TOMPKINS, 1988; AVIS; BARBER, 1994; CHENEY; ISAACSON, 1991a). Hua et al. (HUA; WEBSTER; TOMPKINS, 1988) também sugeriram um método cruzado, onde a corrente é injetada entre pares de eletrodos que estão mais afastados um do outro. Este método obtém uma distribuição de corrente mais uniforme. Em todos os padrões, o par de eletrodos excitado é alternado sucessivamente em torno da seção transversal do corpo até se obter um conjunto de observações (padrões de corrente) linearmente independentes, determinado pelo número de eletrodos utilizados (SEAGAR; BARBER; BROWN, 1987; TANG et al., 2002). Assim, o aumento da resolução da imagem na TIE está relacionado com a quantidade de informação de potenciais elétricos obtidos no contorno do domínio submetido à tomografia, portanto para aumentar a resolução é conveniente aumentar o número de eletrodos posicionados ao redor do corpo (FUCKS, 1991; TANG et al., 2002). Isso está ilustrado na Fig. 2, em que são apresentadas 4 padrões de corrente distintos, seguindo um padrão “pula três”, para 32 eletrodos posicionados na fronteira do domínio. Observa-se

Geralmente, na TIE, aplica-se corrente elétrica alternada (AC) (LAWRENCE, 1996). Em aplicações médicas, a corrente contínua (DC) provoca corrosão no eletrodo devido à ação eletrolítica no contato do eletrodo com a pele do paciente, podendo causar a formação de úlceras na pele na região do contato (BROWN, 1983). A faixa de intensidade e frequência de corrente alternada aplicada na TIE varia conforme a sua aplicação. Por exemplo, nas aplicações para a área médica, a intensidade e frequência variam de 1 A a 10 mA e de 10 kHz a 1 MHz, respectivamente (NEWELL et al., 1996).

Conhecendo-se os valores de potenciais elétricos, medidos nos eletrodos, um computador acoplado ao hardware da TIE recebe os valores de potenciais lidos pelo sistema de aquisição de dados e através de um algoritmo de reconstrução procede a estimação da distribuição de condutividade no interior da seção do corpo, associando uma imagem

Figura 2 – Ilustração de 4 dos 32 padrões de corrente. Os triângulos, na borda, representam os 32 eletrodos. (a) Primeiro padrão de corrente, com injeção de corrente nos eletrodos 1 e 5. (b) Segundo padrão de corrente, com injeção de corrente nos eletrodos 2 e 6. (c) Terceiro padrão de corrente, com injeção de corrente nos eletrodos 3 e 7. (d) Trigésimo segundo, e último, padrão de corrente, com injeção de corrente nos eletrodos 32 e 4.

gráfica (GISSER; ISAACSON; NEWELL, 1988; RIGAUD et al., 1993). Pode-se dizer que a imagem obtida é uma distribuição de condutividades (ou resistividades) que minimiza uma particular “função-erro”, inerente ao algoritmo de reconstrução utilizado (CHENEY; ISAACSON, 1991b; COOK et al., 1994).

Aspectos teóricos da obtenção de imagem através da TIE mostram que é possível estimar uma distribuição uniforme de condutividades oriundas de materiais isotrópicos a partir de medidas de potenciais, desde que essas medidas sejam infinitamente precisas e obtidas em todos os pontos do contorno da seção do corpo (SYLVESTER; UHLMANN, 1987). Porém, na prática essas condições não são atingíveis devido às limitações em se medir de forma precisa os valores de potenciais através do aparato eletrônico utilizado na TIE, de onde se extrai um número finito de medições de potenciais através de um número determinado de eletrodos em torno da seção do corpo (TANG et al., 2002). Na literatura encontram-se diversos algoritmos de reconstrução, com diferentes estratégias adotadas para lidar com essas limitações da TIE (YORKEY; WEBSTER; TOMPKINS, 1987).

Aplicada à área médica, a TIE é uma técnica de monitoração em tecidos biológicos que, basicamente, consiste na obtenção da imagem de uma seção transversal do corpo do paciente, onde cada ponto na imagem representa a sua resistividade elétrica estimada, a partir de injeções de corrente elétrica em eletrodos posicionados ao redor dessa seção (CHENEY; ISAACSON; NEWELL, 1999). A seguir, é mostrado o histórico da TIE aplicada à área médica, bem como suas possíveis aplicações clínicas.

Os primeiros estudos e formulações teóricas para implementação de algoritmos e hardwares de reconstrução de imagem que fundamentaram a TIE em aplicações clínicas surgiram na década de 80 na Universidade de Sheffield (Inglaterra). Pesquisadores desta universidade estudaram a técnica de TIE, e descreveram as razões do interesse para a aplicação clínica da TIE como, por exemplo, no monitoramento de funções pulmonares, cardíacas e gástricas, e chamaram a atenção para futuros desenvolvimentos desta técnica de tomografia (BARBER; BROWN, 1984). Eles são praticamente os pioneiros na formulação de teorias e obtenção de dados que baseiam a TIE até o presente momento na área médica. É deles o desenvolvimento e a patente do primeiro aparelho comercial de TIE. Este aparelho é baseado no método de “backprojection” e possui um sistema denominado APT (“Applied Potencial Tomography”) (BARBER; BROWN, 1984), que é composto por uma fonte simples de corrente elétrica, 16 eletrodos e utiliza o padrão subjacente para aplicação da corrente elétrica. Embora este sistema APT permita apenas obter imagens de baixa resolução, ele tem sido utilizado em estudos de vários procedimentos médicos como o monitoramento do fluxo de sangue no tórax e problemas de colapso no pulmão (HOLDER, 1993), por exemplo.

Na década de 90, diversos pesquisadores desenvolveram outros projetos e instrumentações experimentais que avaliam o método de “backprojection” para a TIE (SANTOSA;

VOGELIUS, 1990) e descreveram a importância da obtenção dos resultados experimentais nos denominados fantasmas (PAULSON; BRECKON; PIDCOCK, 1992). Por exemplo, pesquisadores do Rensselaer Polytechnic Institute (EUA) projetaram e construíram um outro aparelho de TIE com um sistema denominado de ACT (“Adaptive Current Tomography”) (CHENEY; ISAACSON; NEWELL, 1999), que é capaz de obter 20 imagens estáticas por segundo e utiliza 32 eletrodos. Este aparelho utiliza um método rápido (relativo) de reconstrução de imagem baseado no algoritmo de Newton de Uma Iteração (NOSER) (CHENEY et al., 1990; HYARIC, 2000).

No Brasil, o desenvolvimento de algoritmos e hardwares para a TIE foi feito por pesquisadores da Universidade de São Paulo num projeto temático da FAPESP que estudou as técnicas de reconstrução de imagem aplicadas a um aparelho de TIE para monitorar de forma precisa a ventilação artificial do pulmão (AMATO et al., 1998). Nesse projeto, alguns trabalhos foram publicados, como os artigos de Vanegas e Trigo et al. (VANEGAS, 2002; TRIGO; LIMA; AMATO, 2003; TRIGO; LIMA; AMATO, 2004) que propõem a aplicação de um algoritmo de reconstrução que utiliza o filtro de Kalman estendido (TRIGO, 2005) para a estimação da distribuição de condutividades absolutas na TIE, a dissertação de Molina (MOLINA, 2002; MOLINA et al., 2003) que estuda o refinamento iterativo da malha de elementos finitos do domínio discreto para reduzir os erros numéricos no filtro de Kalman estendido, a dissertação de Miranda (MIRANDA, 2003) que utiliza o método iterativo de Newton-Raphson para estimar a distribuição de condutividades. Também no sentido de se utilizar abordagens baseadas em métodos de otimização, pode-se citar a tese de Lima (LIMA; SILVA; GONZALES-LIMA, 2003; MOTTA, 2005; LIMA, 2006) que aplicou otimização topológica e programação linear sequencial para resolver o problema da TIE, enquanto (HERRERA, 2007) utiliza o recozimento simulado com o mesmo objetivo.

2.3 Imagem Absoluta e Imagem Relativa

Na TIE, o imageamento estático se refere a obter uma distribuição de propriedades (impeditividade ou resistividade) que não se altera ao longo do tempo. Neste caso, a precisão dos valores das tensões medidas e calculadas na TIE é de fundamental importância para garantir uma imagem confiável. Para este imageamento, torna-se importante obter valores absolutos para as grandezas medidas, pois permitem representar a geometria de forma completa com uma única imagem. No caso, obter os valores absolutos de resistividade na TIE pode ser útil em aplicações médicas, tais como o monitoramento do pulmão. A variação da resistividade no pulmão é bastante significativa durante a respiração, portanto conhecendo os valores absolutos dessa propriedade em cada ponto da imagem obtida, o médico pode identificar qual região do pulmão contém mais ou menos ar e se existe alguma região com líquido em excesso, por exemplo.

A obtenção de imagens relativa, ou de diferenças, na TIE está mais relacionada com a distribuição de condutividades estimadas a partir de dois instantes diferentes de tempo – imageamento dinâmico, considerando um deles como referencial. Essas imagens são úteis para monitorar o fluxo de um processo no tempo. Por exemplo, o monitoramento da função pulmonar durante a respiração. Neste caso, o médico pode usar as imagens de diferenças para acompanhar a evolução de alguma anomalia que ocorra no pulmão. Como esse monitoramento ocorre praticamente em tempo real, o algoritmo utilizado para gerar a imagem deve ser rápido, portanto o uso de métodos não-iterativos, neste caso, é a melhor opção.

É importante não associar diretamente imagens absolutas e estáticas, assim como imagens relativas e dinâmicas. Imagens absolutas são imagens que possuem valores absolutos de uma distribuição de propriedades, diferentemente de imagens relativas, que tomam outra imagem, geralmente absoluta, como referência. Imagens estáticas e dinâmicas estão relacionadas ao imageamento – realizado em um único instante para obter a distribuição instantânea ou realizado ao longo do tempo para se acompanhar a variação da distribuição de propriedades de interesse.

Na literatura, uma parte dos estudos em TIE concentram-se no desenvolvimento de métodos para a obtenção de imagens relativas, e outra parte para obtenção de imagens estáticas, as quais vêm sendo investigadas utilizando-se diversos métodos originais, tais como os algoritmos baseados no método de Newton-Raphson (CHENEY et al., 1990; YORKEY; WEBSTER; TOMPKINS, 1987), no filtro de Kalman (TRIGO; LIMA; AMATO, 2004; VAUHKONEN; KARJAKAINEN; KAIPIO, 1998), no algoritmo genético (OLMI; BINI; PRIORI, 2000) e na otimização topológica (LIMA, 2006).

2.4 A TIE e seus Algoritmos de Reconstrução

Desde o advento da TIE, vários algoritmos de reconstrução de imagem são aplicados para resolver o seu problema inverso e não linear, sendo que um comparativo entre alguns deles pode ser visto em (YORKEY; WEBSTER; TOMPKINS, 1987). Os algoritmos de reconstrução para a TIE podem ser divididos em duas classes: algoritmos não iterativos e algoritmos iterativos (LEWITT, 1983). Os algoritmos não iterativos baseiam-se em aproximações lineares do seu problema inverso e na hipótese de distribuição homogênea de condutividade. Embora desprezem o efeito da não linearidade, esses algoritmos são bastante atrativos, pois lidam com formulações matemáticas não muito complexas e são muito rápidos computacionalmente, obtendo dezenas de imagens por segundo. Já, os algoritmos iterativos convertem o problema não linear da TIE numa série de problemas lineares menores e por meio da solução sucessiva desses problemas menores buscam a solução do problema original (CENSOR, 1983).

No processo iterativo, inicialmente assume-se uma distribuição de impeditividades e para um valor de corrente elétrica conhecida resolve-se as equações de Poisson do domínio condutivo em questão, determinando assim a distribuição de potenciais elétrico deste domínio. Os valores de potenciais encontrados no contorno do domínio são comparados com valores de potenciais elétricos medidos previamente. A diferença entre esses potenciais (medidos e calculados) é utilizada para orientar o algoritmo a buscar a cada iteração, sucessivamente, a distribuição de potenciais elétricos calculada que mais se aproxima da distribuição de potenciais real do domínio. Cada algoritmo iterativo se diferencia um do outro pela maneira de ajustar os valores de condutividade estimados a cada iteração (BARBER, 1989). Os algoritmos iterativos são mais sensíveis a ruídos nos dados fornecidos e a imagem é obtida num tempo maior, contudo permitem estimar valores absolutos de condutividades com uma precisão maior. Neste trabalho de pesquisa, os algoritmos aplicados para a obtenção de imagens na TIE são baseados no recozimento simulado.

Na literatura já existem estudos referentes à implementação do recozimento simulado na TIE (HERRERA, 2007), neles obtiveram-se bons resultados preliminares para testes com dados simulados, porém discute-se o tempo de processamento, que é grande devido ao número de graus de liberdade do problema. Este assunto será melhor abordado no capítulo 4.

2.5 O Problema Direto da TIE

O problema inverso é formulado como dadas correntes injetadas $\mathbf{J}$ e os potenciais sobre os eletrodos de medição ϕ_m , encontrar a distribuição de condutividade elétrica σ sobre Ω . A distribuição de condutividade pode ser obtida através da equação elíptica (2.1), aplicando-se as condições de contorno dadas pelas correntes J na fronteira do domínio, como definido por (2.2). A formulação matemática completa é apresentada no apêndice A.

$$\nabla \cdot (\sigma \nabla \phi) = 0. \quad (2.1)$$

$$\sigma \frac{\partial \phi}{\partial \hat{n}} = J. \quad (2.2)$$

Para um domínio irregular e meio isotrópico, soluções analíticas para a equação de Laplace (2.1) com condição de contorno (2.2) são desconhecidas; então, as equações diferenciais parciais são aproximadas pelo Método dos Elementos Finitos (MEF) (CASTRO SOBRINHO, 2006; BATHE, 1995), o domínio é discretizado como elementos lineares triangulares com condutividade constante em ambos os problemas, direto e inverso, e

resolvido numericamente. O princípio dos potenciais virtuais associado à equação de Laplace fornece a matriz local do elemento.

Quando a matriz local do elemento for escrita em termos de coordenadas globais da malha, a matrix global de condutividade (TRIGO; LIMA; AMATO, 2004) que inclui o modelo de eletrodo, é obtida. A relação

$$\mathbf{K} \times \Phi = \mathbf{C} \quad (2.3)$$

é válida, onde $\mathbf{K}(\sigma) \in \mathbb{R}^{s \times s}$ é a matriz de condutância calculada em uma distribuição particular σ_p , Φ é a matriz contendo os potenciais nodais correspondentes a cada padrão de corrente aplicado, e $\mathbf{C}$ representa p padrões de corrente linearmente independentes. $\mathbf{C}$ é dado por

$$\mathbf{C} = \begin{bmatrix} -I & -I & -I & \cdots & -I \\ I & 0 & 0 & \cdots & 0 \\ 0 & I & 0 & \cdots & 0 \\ 0 & 0 & I & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & I \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 0 \end{bmatrix}_{s \times p} \quad (2.4)$$

onde I representa a intensidade da corrente injetada.

2.6 Malhas para o Modelo de Elementos Finitos (MEF)

Neste trabalho, é proposta uma nova parametrização para criar malhas para o MEF. As malhas criadas são divididas em duas regiões, interna e anel e externo, como explicado a seguir. O objetivo principal é estudar a influência de cada região e a influência do refinamento em cada seção no erro induzido pela malha. Um refinamento maior da malha eleva o custo computacional, mas não há uma relação direta na redução do erro do modelo. Isto será melhor abordado na seção 2.7.

2.6.1 Criação da malha

A criação de malha que divide o domínio em duas regiões principais: anel externo, que inclui os eletrodos, e uma área circular central. A definição das duas regiões permite a criação de malhas com densidades distintas. O refinamento do anel externo está diretamente conectado ao modelo do eletrodo. Se mais elementos são desejados no anel externo, então o modelo do eletrodo deve permitir um maior número de elementos também. Tanto os

Figura 3 – (a) Uma malha com 30.216 nós, com alta densidade de nós na região circular interna e eletrodos simples. (b) Detalhe da diferença no refinamento das regiões interna e anel externo, para a malha mostrada em (a).

Figura 4 – (a) Uma malha com 29.074 nós, com eletrodos com alta densidade de nós e região circular interna com baixa densidade. (b) Detalhe da diferença no refinamento das regiões interna e anel externo, para a malha mostrada em (a).

Figura 5 – (a) Uma malha com 29.517 nós, com densidade média de nós nas regiões circular interna e do anel externo (e eletrodos). (b) Detalhe da diferença no refinamento das regiões interna e anel externo, para a malha mostrada em (a).

eletrodos quanto a região circular interna possuem níveis de refinamento independentes e controláveis. O anel externo, em si, funciona como uma região de transição entre os eletrodos e a região mais interna.

O nível de refinamento dos eletrodos é controlado pelo número de nós na sua base. A Fig. 3(b) mostra um eletrodo com apenas três nós na sua base. Por outro lado, a Fig. 4(b) mostra um eletrodo com base altamente refinada.

O nível de refinamento da região circular interna é determinado pelo número de nós no seu contorno, que pode ser definido de forma arbitrária. Na Fig. 3(a), é mostrada uma malha com alta densidade na região interna, mas baixa densidade nos eletrodos. Na Fig. 4(a), é mostrada uma malha com baixa densidade na região interna, mas alta densidade nos eletrodos. Na Fig. 4(b), é possível ver claramente a diferença nas densidades das duas regiões, e também a diferença com a situação da outra malha (vide Fig. 3(b)).

Uma terceira malha, com nós mais uniformemente distribuídos entre todas as regiões, é mostrada na Fig. 5. Todas essas malhas possuem cerca de trinta mil nós, mas a distribuição dos nós entre as duas regiões é diferente e ilustram as diversas possibilidades de densidade de nós para a criação da malha.

2.6.2 Determinação da Matriz de Condutividade

Para elementos triangulares, pode-se definir a função de interpolação

$$\phi^e(x, y) = a + bx + cy \quad (2.5)$$

onde a , b e c são constantes desconhecidas e que devem ser determinadas. Portanto, o potencial elétrico nos nós 1, 2 e 3 do elemento triangular (vide Fig. 6) se torna

$$\begin{bmatrix} \phi_1^e \\ \phi_2^e \\ \phi_3^e \end{bmatrix}^T = \begin{bmatrix} a \\ b \\ c \end{bmatrix}^T \times \begin{bmatrix} 1 & 1 & 1 \\ x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{bmatrix}.$$

E, após algumas operações algébricas, (2.5) se torna

$$\phi^e(x, y) = \frac{1}{2\|\mathbf{A}\|} \begin{bmatrix} \phi_1^e \\ \phi_2^e \\ \phi_3^e \end{bmatrix}^T \begin{bmatrix} x_2y_3 - x_3y_2 & y_{23} & x_{32} \\ x_3y_1 - x_1y_3 & y_{31} & x_{31} \\ x_1y_2 - x_2y_1 & y_{12} & x_{21} \end{bmatrix} \begin{bmatrix} 1 \\ x \\ y \end{bmatrix}$$

onde $x_{ij} = x_i - x_j$ e $y_{ij} = y_i - y_j$, ϕ_i^e é o potencial elétrico em um dado nó e $\|\mathbf{A}\|$ é a área do elemento. E, finalmente, os elementos da matriz, dado um simples elemento triangular, são definidos como

$$k_{11}^e = \frac{\sigma \times h}{4\|\mathbf{A}\|} [y_{23}^2 + x_{32}^2], k_{22}^e = \frac{\sigma \times h}{4\|\mathbf{A}\|} [y_{31}^2 + x_{31}^2]$$

Figura 6 – Elemento isoparamétrico triangular, com nós numerados no sentido anti-horário.

Figura 7 – Modelo de eletrodo, com 3 nós na base.

$$\begin{aligned}
 k_{33}^e &= \frac{\sigma \times h}{4\|\mathbf{A}\|} [y_{12}^2 + x_{21}^2] \\
 k_{12}^e = k_{21}^e &= \frac{\sigma \times h}{4\|\mathbf{A}\|} [y_{23} \times y_{31} + x_{32} \times x_{13}] \\
 k_{13}^e = k_{31}^e &= \frac{\sigma \times h}{4\|\mathbf{A}\|} [y_{23} \times y_{12} + x_{32} \times x_{21}] \\
 k_{32}^e = k_{23}^e &= \frac{\sigma \times h}{4\|\mathbf{A}\|} [y_{31} \times y_{12} + x_{13} \times x_{21}]
 \end{aligned}$$

A matriz global de condutividade $\mathbf{K}$ pode ser definida como $\mathbf{K} = \sum \mathbf{k}^e$. Esta matriz é simétrica e singular.

2.6.3 Modelo de Eletrodo

O eletrodo é modelado como mostrado na Fig. 7. Uma placa metálica, em contato com a superior do modelo, curto-circuita os nós A , B e C , deixando-os com o mesmo potencial elétrico. O número de elementos triangulares no eletrodo é controlado por um dos parâmetros de refinamento mencionados acima. A partir de (2.3), para a malha mostrada

Figura 8 – Modelo de eletrodo simplificado, com base de 3 nós.

na Fig. 7, o seguinte sistema linear é definido

$$\begin{bmatrix} k_{AA} & k_{AB} & 0 & k_{A1} & k_{A2} & 0 \\ k_{BA} & k_{BB} & k_{BC} & 0 & k_{B2} & k_{B3} \\ 0 & k_{CB} & k_{CC} & 0 & 0 & k_{C3} \\ k_{1A} & 0 & 0 & k_{11} & k_{12} & 0 \\ k_{2A} & k_{2B} & 0 & k_{21} & k_{22} & k_{23} \\ 0 & k_{3B} & k_{3C} & 0 & k_{32} & k_{33} \end{bmatrix} \begin{bmatrix} \phi_A \\ \phi_B \\ \phi_C \\ \phi_1 \\ \phi_2 \\ \phi_3 \end{bmatrix} = \begin{bmatrix} J_A \\ J_B \\ J_C \\ J_1 \\ J_2 \\ J_3 \end{bmatrix} \quad (2.6)$$

Considerando $\phi_A = \phi_B = \phi_C = \phi_b$, obtém-se

$$\begin{aligned} \phi_b(k_{AA} + k_{AB}) + \phi_1 k_{A1} + \phi_2 k_{A2} &= J_A \\ \phi_b(k_{BA} + k_{BB} + k_{BC}) + \phi_2 k_{B2} + \phi_3 k_{B3} &= J_B \\ \phi_b(k_{CB} + k_{CC}) + \phi_3 k_{C3} &= J_C \\ \phi_b k_{1A} + \phi_1 k_{11} + \phi_2 k_{12} &= J_1 \\ \phi_b(k_{2A} + k_{2B}) + \phi_1 k_{21} + \phi_2 k_{22} + \phi_3 k_{23} &= J_2 \\ \phi_b(k_{3B} + k_{3C}) + \phi_2 k_{32} + \phi_3 k_{33} &= J_3 \end{aligned}$$

Portanto, (2.6) se torna

$$\begin{bmatrix} k_{bb} & k_{b1} & k_{b2} & k_{b3} \\ k_{1b} & k_{11} & k_{12} & 0 \\ k_{2b} & k_{21} & k_{22} & k_{23} \\ k_{3b} & 0 & k_{32} & k_{33} \end{bmatrix} \begin{bmatrix} \phi_b \\ \phi_1 \\ \phi_2 \\ \phi_3 \end{bmatrix} = \begin{bmatrix} J_A + J_B + J_C \\ J_1 \\ J_2 \\ J_3 \end{bmatrix} \quad (2.7)$$

onde

$$\begin{aligned} k_{bb} &= k_{AA} + 2k_{AB} + k_{BB} + 2k_{BC} + k_{CC} \\ k_{b1} &= k_{1A}, \quad k_{b2} = k_{A2} + k_{B2}, \quad k_{b3} = k_{C3} + k_{B3} \end{aligned}$$

Portanto, o modelo de eletrodo pode ser simplificado para o modelo mostrado na Fig. 8.

Figura 9 – Modelo de eletrodo genérico.

Figura 10 – Uma subdivisão do eletrodo.

2.6.4 Modelo de Eletrodo com Número de Elementos Variável

O eletrodo com número variável de elementos é modelado como mostrado na Fig. 9. Da mesma forma que no eletrodo simples, há uma placa metálica em contato com a superfície superior, curto-circuitando todos os nós nesta superfície – A, B, C, D e assim por diante, que passam a ter o mesmo potencial elétrico. Considere uma subdivisão desse eletrodo, como mostrada na Fig. 10. Um eletrodo completo é formado por diversas dessas subdivisões, e a quantidade destas células é controlada pelo parâmetro de refinamento do eletrodo. Esta subdivisão contém dois elementos triangulares, e o seguinte sistema linear é definido

$$\begin{bmatrix} k_{MM} & k_{MN} & k_{Mi} & k_{Mj} \\ k_{NM} & k_{NN} & 0 & k_{Nj} \\ k_{iM} & 0 & k_{ii} & k_{ij} \\ k_{jM} & k_{jN} & k_{ji} & k_{jj} \end{bmatrix} \begin{bmatrix} \phi_M \\ \phi_N \\ \phi_i \\ \phi_j \end{bmatrix} = \begin{bmatrix} J_M \\ J_N \\ J_i \\ J_j \end{bmatrix} \quad (2.8)$$

Considerando $\phi_M = \phi_N = \phi_b$ e que $k_{ab} = k_{ba}$, (2.8) se torna

$$\begin{bmatrix} k_{bb} & k_{bi} & k_{bj} \\ k_{bi} & k_{ii} & k_{ij} \\ k_{bj} & k_{ij} & k_{jj} \end{bmatrix} \begin{bmatrix} \phi_b \\ \phi_i \\ \phi_j \end{bmatrix} = \begin{bmatrix} J_M + J_N \\ J_i \\ J_j \end{bmatrix} \quad (2.9)$$

onde

$$k_{bb} = k_{MM} + 2k_{MN} + k_{NN}, \quad k_{bi} = k_{Mi}, \quad k_{bj} = k_{Mj} + k_{Nj}.$$

Considerando que todos os nós superiores da Fig. 9 – A, B, C e assim por diante – possuem o mesmo potencial elétrico, pode-se reduzir o número de variáveis, reduzindo o tamanho

Figura 11 – Modelo de eletrodo genérico simplificado.

da matriz e, portanto, reduzindo o custo computacional. Portanto, o modelo de eletrodo genérico pode ser simplificado para o modelo mostrado na Fig. 11.

2.7 Estudo do Erro das Malhas

Na TIE, o problema inverso tem sua solução diretamente afetada pela insuficiência no refinamento da malha utilizada no problema direto. Um fator de correção pode ser utilizado para reduzir o efeito do refinamento insuficiente da malha e simultaneamente melhorar a qualidade da solução do problema inverso.

Silva (SILVA, 2012) propôs uma formulação seguindo a teoria encontrada em (KAIPIO; SOMERSALO, 2004). Considere solucionar o problema direto utilizando uma malha extremamente refinada. O resultado é um mapa de potenciais elétricos em cada nó da malha extremamente refinada, e será considerado preciso. Em outras palavras, esta malha pode ser considerada tão refinada que erros devido à discretização podem ser desconsiderados, e esta malha será chamada como “modelo de observação”. Pode-se usar

$$\Phi = f(\rho)$$

onde Φ são os potenciais elétricos precisos, calculados com o modelo de observação, e ρ define o estado em que o experimento ocorreu. Agora, considere $f(\rho, h)$ representando o modelo de observação preciso, dependendo do estado ρ e do nível de refinamento da malha, h . Quanto menor for h , mais refinada é a malha. Finalmente, considere um erro aditivo Φ_ϵ , produzido por fenômenos não modelados. Agora, o potencial elétrico Φ se torna

$$\Phi = f(\rho, h) + \Phi_\epsilon \quad (2.10)$$

Considere um “modelo de observação reduzido”, definido por um estado dimensionalmente reduzido ρ^r e uma malha mais grossa com um maior parâmetro de discretização h^r . Os erros produzidos por fenômenos não modelados se tornam Φ_ϵ^r e, portanto, os potenciais elétricos calculados para este modelo reduzido se torna

$$\Phi^r = f(\rho^r, h^r) + \Phi_\epsilon^r.$$

Manipulando (2.10), ao se somar e subtrair Φ^r , obtém-se

$$\begin{aligned}\Phi &= f(\rho, h) + \Phi_\epsilon + \Phi^r - \Phi^r \\ &= f(\rho, h) + \Phi_\epsilon + (f(\rho^r, h^r) + \Phi_\epsilon^r) - (f(\rho^r, h^r) + \Phi_\epsilon^r) \\ &= f(\rho^r, h^r) + (f(\rho, h) - f(\rho^r, h^r)) + \Phi_\epsilon\end{aligned}$$

e usando $\nu = (f(\rho, h) - f(\rho^r, h^r)) + \Phi_\epsilon$, obtém-se

$$\Phi = f(\rho^r, h^r) + \nu. \quad (2.11)$$

E, portanto, os potenciais elétricos precisos podem ser calculados a partir de um modelo reduzido $f(\rho^r, h^r)$ utilizando uma malha mais grossa, e usando ν como fator de correção de erro de discretização. Este fator de correção ν pode ser calculado solucionando-se o problema direto com uma malha extremamente refinada (modelo de observação) e malhas mais grossas (modelos reduzidos). Os erros devido a fenômenos não modelados serão considerados como zero, para simplificação.

2.7.1 Análise do Erro

Como explicado acima, o modelo de malhas proposto neste trabalho possui duas regiões: um anel externo, contendo os eletrodos, e uma região circular interna. A Fig. 12 mostra a variação do número de nós como domínio do espaço de malhas estudadas. No eixo- X , o número de nós no anel externo é mostrado. No eixo- Y , o número de nós na região mais interna. A cor indica o número total de nós.

Um total de cerca de 200 malhas foram estudadas, com um número total de nós nas malhas variando de 606 a 186.477. A razão do número de nós entre o anel externo e a região interna varia de 0,08 a 46,29. Um malha adicional, mais refinada, com 459.624 nós foi criada para ser utilizada como referência, como descrito pela teoria de discretização do erro. O domínio é considerado como sendo uma solução salina e a condutividade do eletrodo é cerca de 20 vezes a condutividade desta solução.

A corrente é aplicada no modelo em padrões “pula três”, isto é, a corrente é aplicada em eletrodos separados por outros três eletrodos. A distribuição do potencial elétrico determinado pelo problema direto, para a malha de referência, é mostrada na Fig. 13. O potencial elétrico medido em cada um dos 31 eletrodos é mostrado na Fig. 14. É importante observar que a corrente é aplicada no primeiro eletrodo, saindo pelo quinto. O eletrodo 32 não é apresentado, pois é utilizado como nó terra. Isto, inclusive, é feito em todas as malhas – em todas elas, o eletrodo 32 é o nó terra. O terra é um valor de potencial arbitrário, escolhido para equalizar os potenciais entre as diferentes malhas. Pode ser escolhido de forma que a diferença entre os potenciais nos eletrodos do modelo de observação e dos modelos reduzidos seja diminuída. A equalização é representada como a

Figura 12 – Espaço de malhas exploradas, considerando o número de nós no anel externo e nas regiões circulares internas. N_{NER} = Número de nós no anel externo, N_{NIC} = Número de nós na região circular interna. A cor indica o número total de nós.

Figura 13 – Distribuição do potencial elétrico para a malha de referência.

minimização de ϕ_ϵ , formulada como

$$\min_{\phi_\epsilon} \left\{ \sum_{i=1}^{32} (\phi_i - \phi_i^r - \phi_\epsilon)^2 \right\} = \sum_{i=1}^{32} \frac{\partial (\phi_i - \phi_i^r - \phi_\epsilon)^2}{\partial \phi_\epsilon} = 0.$$

Para todas as malhas criadas, o módulo de ν (ver (2.11) – erro quadrático entre os potenciais nos eletrodos do modelo de observação – malha de referência – e do modelo reduzido – malha grossa) foi calculado e comparado com o número total de nós, e a razão entre os números de nós no anel externo e na região circular interna, vide Fig. 15. Como

Figura 14 – Potenciais elétricos nos eletrodos, para a malha de referência.

Figura 15 – Gráfico mostrando a relação entre o número total de nós da malha (horizontal) e a razão entre os números de nós nas regiões do anel externo e na região circular interna (vertical). Cor indica o módulo de ν (mV).

Figura 16 – Gráfico mostrando os erros nos potenciais (em cor) nos eletrodos ativos – eletrodos em que são aplicadas correntes – associados ao número de nós no anel externo (vertical) e na região circular interna (horizontal).

pode-se observar, maiores razões indicam menores erros, ainda que isto não possa ser afirmado de maneira conclusiva.

Em uma comparação entre as três malhas apresentadas nas Figs. 3, 4 e 5, o erro quadrático foi, respectivamente, 43,61, 21,3 e 38,33. A malha mais refinada no anel externo possui o menor erro, enquanto que a malha mais refinada no anel interno possui o maior erro. Estas três malhas indicam que é mais importante analisar cada região separadamente, ao invés de uma análise global direta.

A Fig. 16 mostra os erros nos potenciais entre os eletrodos ativos, eletrodos onde a corrente é injetada em cada malha, organizados de acordo com o número de nós no anel externo e na região circular interna. A cor representa a variação do valor do erro. É claro que, na média, o número de nós na camada externa reduz o erro no potencial entre eletrodos ativos. A Fig. 17 mostra um resultado semelhante para os eletrodos passivos – demais eletrodos. O resultado mostra uma influência bastante ruidosa onde praticamente nenhuma tendência pode ser definida.

Assim, baseando-se nos resultados destas famílias de malhas, é possível concluir que o anel externo influencia o erro nos potenciais dos eletrodos ativos, isto é, aqueles em que se aplica corrente. Portanto, a escolha apropriada – com o refinamento principalmente no anel exterior – para os níveis de refinamento dos eletrodos e da região circular interna pode efetivamente reduzir o número de nós necessários para se atingir a qualidade desejada. Com a redução do número de nós, o custo computacional também é reduzido.

Figura 17 – Gráfico mostrando os erros nos potenciais (em cor) nos eletrodos passivos – eletrodos em que não são aplicadas correntes – associados ao número de nós no anel externo (vertical) e na região circular interna (horizontal).

2.8 O Algoritmo de Gradientes Conjugados

O algoritmo de Gradientes Conjugados é um método de solução numérica para sistemas lineares na forma

$$Ax - b = 0 \quad (2.12)$$

quando A é uma matriz quadrada simétrica positiva definida. Assim, ele é particularmente apropriado para resolver sistemas como os gerados pelo MEF descrito na seção 2.5.

Ele é um processo iterativo que parte de uma solução inicial x_0 e a cada iteração se aproxima da solução exata. Em aritmética exata, pode-se mostrar que o algoritmo encontra a solução exata em no máximo n iterações, onde n é o posto de A (SHEWCHUK, 1994). Em geral, é computacionalmente inviável trabalhar com aritmética exata. Ainda assim, o método é numericamente estável e capaz de encontrar soluções numéricas aceitáveis em aritmética de precisão finita (MEURANT, 2006, cap.8).

As iterações do algoritmo se dão como se segue (adaptado de (SHEWCHUK, 1994))

$$r_0 = b - Ax_0 \quad (2.13)$$

$$d_0 = r_0 \quad (2.14)$$

$$\alpha_i = \frac{r_i^T r_i}{d_i^T A d_i} \quad (2.15)$$

$$x_{i+1} = x_i + \alpha_i d_i \quad (2.16)$$

$$r_{i+1} = r_i - \alpha_i A d_i \quad (2.17)$$

$$\beta_{i+1} = \frac{r_{i+1}^T r_{i+1}}{r_i^T r_i} \quad (2.18)$$

$$d_{i+1} = r_{i+1} + \beta_{i+1} d_i \quad (2.19)$$

Como se observa, as operações mais computacionalmente intensivas em uma iteração são o produto matriz $\times$ vetor em (2.15), que ele pode ser reaproveitado em (2.17).

2.8.1 Convergência

Seja ϵ_k o erro na k -ésima iteração, e $\|\epsilon_k\|_A = \epsilon_k^T A \epsilon_k$ a sua A -norma. Então pode ser mostrado (vide por exemplo (MEURANT, 2006, p.78) ou (SHEWCHUK, 1994)) que

$$\|\epsilon_k\|_A \leq 2 \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right) \|\epsilon_0\|_A \quad (2.20)$$

onde $\kappa = \lambda_{\max}/\lambda_{\min}$ é o *Número de Condicionamento* de A , e é definido como a razão entre o maior autovalor de A e o menor autovalor. Pelo teorema espectral, como A é simétrica e positiva definida, todos os seus autovalores são positivos (WILF, 1962, p.28).

Assim, espera-se que a convergência do algoritmo dos Gradientes Conjugados seja mais lenta para matrizes com número de condicionamento maior.

2.8.2 Pré-Condicionamento

Como visto na seção 2.8.1, o número de condicionamento da matriz A afeta diretamente a taxa de convergência do algoritmo. Seja agora M uma matriz simétrica e positiva definida que aproxime A . Espera-se neste caso que os autovalores de $M^{-1}A$ sejam mais agrupados do que A , e deste modo, o seu número de condicionamento seja menor. Assim, a convergência do algoritmo dos Gradientes Conjugados aplicado ao sistema

$$M^{-1}Ax - M^{-1}b = 0 \quad (2.21)$$

será mais rápida. Nota-se que a solução para o sistema (2.21) é a mesma que a do sistema (2.12).

A questão aqui é que $M^{-1}A$ não é necessariamente simétrica nem positiva definida, invalidando a aplicação dos Gradientes Conjugados ao sistema (2.21).

Modificando-se essa ideia, suponha que seja possível calcular uma decomposição da matriz $M = EE^T$. Neste caso, os autovalores de $M^{-1}A$ e $E^{-1}AE^{-T}$ são os mesmos,

mas esta última matriz é simétrica. Assim, o algoritmo dos Gradientes Conjugados pode ser aplicado diretamente ao sistema

$$E^{-1}AE^{-T}\hat{x} - E^{-1}b = 0 \quad (2.22)$$

notando que $\hat{x} = E^T x$.

O algoritmo pré-condicionado pode ser aplicado diretamente ao sistema (2.22), calculando $\hat{x}$. É geralmente mais eficiente no entanto, aplicar o pré-condicionador diretamente ao cálculo de x . O algoritmo final é (adaptado de (SHEWCHUK, 1994))

$$r_0 = b - Ax_0 \quad (2.23)$$

$$Mz_0 = r_0 \quad (2.24)$$

$$d_0 = z_0 \quad (2.25)$$

$$\alpha_i = \frac{r_i^T z_i}{d_i^T A d_i} \quad (2.26)$$

$$x_{i+1} = x_i + \alpha_i d_i \quad (2.27)$$

$$r_{i+1} = r_i - \alpha_i A d_i \quad (2.28)$$

$$Mz_{i+1} = r_{i+1} \quad (2.29)$$

$$\beta_{i+1} = \frac{z_{i+1}^T r_{i+1}}{z_i^T r_i} \quad (2.30)$$

$$d_{i+1} = z_{i+1} + \beta_{i+1} d_i \quad (2.31)$$

Observa-se que o cálculo de z_{i+1} em (2.29) envolve a solução de um sistema linear. É fundamental portanto que seja possível calcular de forma eficiente $M^{-1}r_{i+1}$, pois esta operação será executada a cada iteração dos Gradientes Conjugados.

A literatura fornece diversos candidatos para o pré-condicionador M . Geralmente, a escolha depende de fatores específicos do problema estudado. Em (MELLO et al., 2008) e em (MARTINS et al., 2011) a decomposição de Choleksy incompleta (KERSHAW, 1978) obteve ganhos dramáticos de desempenho na solução do problema direto da TIE.

2.8.3 Decomposição de Cholesky

A decomposição de Cholesky se aplica à solução de sistemas cujas matrizes são simétricas e positivamente definidas, ou seja, $\vec{v}K\vec{v} > 0$, para todo vetor $\vec{v}$. Ele consiste em decompor a matriz K do sistema em um produto de duas outras matrizes, onde a segunda é a matriz transposta da primeira $L_{ij}^T = L_{ji}$, ou seja

$$K = LL^T.$$

A matriz L é uma matriz com constantes nulas acima da diagonal principal, por isso é denominada matriz inferior. Consequentemente, a matriz L^T é denominada matriz superior. Uma vez efetuada essa decomposição, a solução do sistema de equações pode ser obtida por meio de uma substituição direta e uma retro-substituição. Assim, obtém-se

$$K\Phi = (LL^T)\Phi = L(L^T\Phi) = L\Theta = C.$$

Logo, por substituição direta resolve-se $L\Theta = C$ e, em seguida, obtém-se Φ aplicando a retro-substituição à $L^T\Phi = \Theta$. Note que estes dois últimos passos são idênticos, sendo que o primeiro calcula Θ_i na seguinte ordem

$$\theta_1 = \frac{C_1}{L_{11}}$$

$$\theta_i = \frac{1}{L_{ii}}(C_i - \sum_{k=1}^{i-1} L_{ik}\theta_k).$$

O algoritmo da decomposição de Cholesky incompleta, para o cálculo da matriz triangular inferior L que compõe $K = LL^T$, está apresentado no Algoritmo 1. L_{ii} são as constantes pertencentes à diagonal principal e L_{ij} são as constantes situadas abaixo da diagonal principal da matriz L , ou seja, para $i \neq j$. A implementação padrão está apresentada no Algoritmo 2.

Algorithm 1 Decomposição de Cholesky.

```

for i = 1 to N do
   $L_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} L_{ik}^2}$ 
  for j = i+1 to N do
     $L_{ji} = \frac{1}{L_{ii}} \left( a_{ij} - \sum_{k=1}^{i-1} L_{ik}L_{jk} \right)$ 
  end for
end for

```

2.8.3.1 Decomposição de Cholesky Incompleta

A decomposição de Cholesky incompleta é utilizada como um pré-condicionador para algoritmos como o dos Gradientes Conjugados. A decomposição de Cholesky incompleta é dada por uma matriz triangular inferior esparsa M que é próxima da matriz L determinada na seção anterior. O correspondente pré-condicionador MM^* . A forma

Algorithm 2 Implementação simples da decomposição de Cholesky.

```

 $N = \text{size}(A, 1)$ 
for  $k = 1$  to  $N$  do
   $A(k, k) = \text{sqrt}(A(k, k))$ 
  for  $i = k+1$  to  $N$  do
     $A(i, k) = A(i, k)/A(k, k)$ 
  end for
  for  $j = k+1$  to  $N$  do
    for  $i = j$  to  $N$  do
       $A(i, j) = A(i, j) - A(i, k) * A(j, k)$ 
    end for
  end for
  for  $i = 0$  to  $N$  do
    for  $j = i+1$  to  $N$  do
       $A(i, j) = 0$ 
    end for
  end for
end for

```

adotada neste trabalho utiliza o algoritmo para determinar a decomposição de Cholesky exata, exceto que todo elemento nulo em K é configurado como nulo em M . Isto fornece uma decomposição incompleta que é esparsa na mesma forma que a matriz K .

O algoritmo da decomposição de Cholesky incompleta, para o cálculo da matriz triangular inferior L , está apresentado no Algoritmo 3. A implementação padrão está apresentada no Algoritmo 4.

Algorithm 3 Decomposição de Cholesky incompleta.

```

for  $i = 1$  to  $N$  do
   $L_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} L_{ik}^2}$ 
  for  $j = i+1$  to  $N$  do
    if  $a_{ji} \neq 0$  then
       $L_{ji} = \frac{1}{L_{ii}} \left( a_{ij} - \sum_{k=1}^{i-1} L_{ik} L_{jk} \right)$ 
    end if
  end for
end for

```

2.9 Conclusões

Neste capítulo, a TIE e seus conceitos básicos foram introduzidos, com a apresentação de seu problema direto. O tempo de processamento para o problema direto é diretamente proporcional ao número de elementos existentes na malha que discretiza o domínio. Com a intenção de se determinar a melhor forma de se discretizar o domínio, isto é, reduzir erros não modelados sem o aumento do refinamento, também foi realizado um estudo sobre o impacto do refinamento da malha sobre os potenciais elétricos. Os

Algorithm 4 Implementação simples da decomposição de Cholesky incompleta.

```

 $N = \text{size}(A, 1)$ 
for  $k = 1$  to  $N$  do
   $A(k, k) = \text{sqrt}(A(k, k))$ 
  for  $i = k+1$  to  $N$  do
    if  $A(i, k) \neq 0$  then
       $A(i, k) = A(i, k) / A(k, k)$ 
    end if
  end for
  for  $j = k+1$  to  $N$  do
    for  $i = j$  to  $N$  do
      if  $A(i, j) \neq 0$  then
         $A(i, j) = A(i, j) - A(i, k) * A(j, k)$ 
      end if
    end for
  end for
  for  $i = 0$  to  $N$  do
    for  $j = i+1$  to  $N$  do
       $A(i, j) = 0$ 
    end for
  end for
end for

```

modelos de malha estudados consideram uma divisão do domínio em duas regiões: uma região interna e uma região externa contendo os eletrodos. Foi demonstrado que o número de elementos na região interna não influencia nos erros não modelados, enquanto que o número de elementos no anel externo influencia o erro não modelado presente no eletrodos ativos. Assim, todas as malhas geradas devem um maior nível de refinamento no anel externo.

3 Paralelização do Problema Direto da TIE em GPU

Atualmente, GPUs são unidades de processamento especializadas com enorme capacidade computacional, e altamente paralelizáveis. Sua função é primariamente armazenar conjuntos grandes e complexos de triângulos em um ambiente tridimensional ideal, e manipular diversos desses conforme as necessidades da CPU. Nos últimos anos, as GPUs tornaram-se capazes de realizar os mais incríveis efeitos gráficos, como *blur* para ambientes enevoados, transparências para vidros, janelas e líquidos, reflexos totais e parciais, para efeitos com água. Conforme essas possibilidades cresceram, ao invés de produzir hardwares específicos diferentes para cada efeito, os fabricantes de unidades gráficas passaram a criar unidades gráficas programáveis, que poderiam executar programas arbitrários definidos pelo usuário, para a manipulação de imagens em alguns dos passos chaves da renderização gráfica (NATH et al., 2011).

As CPUs atuais possuem entre quatro e oito núcleos disponíveis, contra centenas de núcleos disponíveis nas GPUs modernas. Tal capacidade levou pesquisadores a utilizar GPUs para solucionar problemas com elevado custo computacional. Áreas como imageamento médico, eletromagnetismo, transcodificação, manipulação de vídeos e problemas com sistemas lineares. A NVIDIA introduziu uma API, chamada CUDA, para computação genérica em GPU (GPGPU - *General Purpose Computing on GPU*), permitindo a utilização de linguagens de programação em alto nível para implementar aplicações de modo massivamente paralelo. O problema de reconstrução da TIE desenvolvido pelo Laboratório de Geometria Computacional utiliza uma biblioteca serial, ainda que eficiente, para as operações de matrizes, chamada Eigen. Tal biblioteca será comparada com a CUDA em algumas das mais utilizadas operações de matrizes.

A arquitetura CUDA utiliza o modelo SIMT (Single Instruction, Multiple Thread), o que significa que todos os núcleos de processamento executam a mesma instrução ao mesmo tempo, de forma parecida à SIMD (Single Instruction, Multiple Data), na qual a mesma operação é aplicada a todo um conjunto de dados simultaneamente. A diferença entre SIMT e SIMD está no fato da SIMT ser capaz de acomodar dados não estruturados e de lidar com divergência de threads (threads rodando simultaneamente executando códigos diferentes). Ambos os casos, porém, geralmente resultam em significativa perda de performance.

A unidade de trabalho básica em GPU é uma thread, executada em um ambiente de memória compartilhada. O número máximo de threads que podem ser executadas é determinado pelo hardware instalado, e o agendador de threads decide quando um grupo de

Figura 18 – Ilustração da arquitetura de memória CUDA.

threads deve ser executado. Um kernel - uma sub-rotina que pode ser invocada a partir do computador (isto é, de um programa tradicional) para ser executada no dispositivo CUDA - deve ser projetado para aproveitar todas as threads disponíveis para realizar sua tarefa. Threads são divididas em blocos, e cada bloco disponibiliza uma memória local rápida, “on-chip” que pode ser compartilhada entre todas as threads do bloco. Dados também podem ser compartilhados entre threads e blocos através da memória global, porém, a memória global é “off-chip” e, portanto, é limitada pela banda de transferência/acesso de memória. Além disso, a memória global está sujeita à regra de coalescência - a memória acessada de forma aleatória apresenta baixo desempenho de acesso e elevada latência. Esta latência no acesso da memória global pode tornar a leitura de um dado 600 vezes mais lenta do que acessar um dado em registrador. A arquitetura da memória CUDA é mostrada na Fig. 18.

A memória global é acessada através de transações de memória de 32, 64 ou 128 bytes. Estas transações precisam ser naturalmente alinhadas, isto é, os endereços de memória acessados precisam ser múltiplos de 32, 64 ou 128 bytes (para o primeiro endereço do bloco) para poderem ser acessados. Isso significa que para acessar endereços de memória não múltiplos de 32, 64 ou 128 bytes, o endereço do primeiro bloco é “arrendondado” para o múltiplo anterior, endereço a partir do qual é contado o tamanho do bloco de acesso. Isso significa que uma leitura de um dado de apenas 4 bytes da memória global (por exemplo, um *int* ou um *float*) irá gerar uma instrução de leitura de 32 bytes, dos quais 28 bytes não serão utilizados, ou seja, a taxa de leitura é dividida por 8.

Quando um warp (um grupo de threads invocadas simultaneamente) solicita acesso à memória global para a leitura de uma posição de um array, por exemplo, o dispositivo CUDA agrupa os acessos à memória global solicitados pelas threads, tentando executar o menor número de transações possível, e diminuir o uso da banda da memória DRAM.

Isso significa que carregar blocos de memória - isto é, carregar dados armazenados em endereços consecutivos da memória - é, portanto, muito mais rápido do que a leitura de dados armazenados aleatoriamente - a regra da coalescência. Para dispositivos modernos, com “compute capability” 2.0 ou superior, o tamanho do warp é de 32 threads. Assim, é muito importante que os dados estejam dispostos de forma que os acessos sejam sempre sequenciais, nas threads de um warp, e em endereços múltiplos de 32, 64 ou 128 bytes.

A diferença entre acessos alinhados - com endereço inicial múltiplo de 32, 64 ou 128 bytes - à memória global pelas 32 threads de um warp e acessos desalinhados - com endereço inicial não-múltiplo de 32, 64 ou 128 bytes - está ilustrada na Fig. 19. A Fig. 19(a) apresenta 32 acessos à memória global, de forma alinhada, realizada pelas threads de um warp. Para esses acesso, é necessária apenas uma única transação de memória (no caso de já estar presente no cache), ou quatro instruções de leitura no caso de ser o primeiro acesso. Já na Fig. 19(b), apenas porque o primeiro endereço de memória acessado não é múltiplo de 32, 64 ou 128 bytes, é necessária uma instrução de acesso adicional à memória global, efetivamente reduzindo a taxa de leitura pela metade no caso de busca no cache - ou de 25% no caso de não estar presente no cache - além de piorar o uso do cache.

Além disso, o dispositivo permite a leitura de estruturas de tamanho 1, 2, 4, 8 ou 16 bytes. Qualquer acesso a dados na memória global se torna uma única instrução se, e somente se, o tamanho da estrutura sendo lida é 1, 2, 4, 8 ou 16 bytes e o endereço é múltiplo desse tamanho. A implicação disso é que ao se trabalhar com estruturas personalizadas, é interessante o preenchimento artificial da estrutura para que sejam acessados mais eficientemente - para estruturas declaradas com o tipo *struct*, pode-se solicitar ao compilador para que faça isso automaticamente.

3.1 Estruturas de Dados para Matrizes Esparsas e Multiplicação matriz esparsa $\times$ vetor

Matrizes de rigidez produzidas pelo modelo MEF em uma reconstrução de imagens TIE são muito esparsas. Há diversas formas para armazenar de maneira eficiente matrizes esparsas, com muito trabalho disponível na literatura analisando suas eficiências - não somente em questão de armazenamento, como também em termos de desempenho computacional. Alguns dos formatos mais comuns incluem COO (*Coordinate Format*), CSR (*Compressed Sparse Row*), BCSR (*Block CSR*), JDS (*Jagged Diagonals Storage*) e ELLPACK.

Formatos típicos para matrizes esparsas se traduzem em sobrecarga de instrução e armazenamento para cada elemento não-nulo, uma vez que a informação para localização dos valores não-nulos precisa ser armazenada. Uma das metas ao selecionar a estrutura de dados mais adequada é minimizar essas sobrecargas. Uma matriz é computacionalmente

Figura 19 – Diferenças entre acessos alinhado e desalinhado à memória global na arquitetura CUDA. (a) Acesso alinhado precisa de quatro (sem cache) ou apenas uma (acesso no cache) operação de leitura de memória para ler 32 valores. (b) No caso de um acesso desalinhado, são necessárias cinco (sem cache) ou dois (acesso no cache) ciclos de leitura.

(a)

(b)

Fonte: [NVIDIA \(2015\)](#), página 215.

Figura 20 – Os elementos não nulos de cada linha são pintados da mesma cor. Cada linha é armazenada como um vetor esparsa, e todas as linhas (isto é, todos os vetores esparsos) são armazenados de forma contígua em v e ci . O vetor p indica onde cada vetor esparsa começa em v e ci .

representada por um vetor unidimensional. Quando a matriz $A_{m \times n}$ é densa, uma abordagem comum é armazenar o elemento (i, j) na posição $v[i \cdot n + j]$. Este mapeamento permite acesso aleatório a qualquer elemento da matriz.

Multiplicação de matrizes esparsas é um dos kernels mais importantes em aplicações científicas e de engenharia. O desafio de desenvolver implementações de alto desempenho de tais kernels é escolher a estrutura de dados e o algoritmo que melhor exploram as propriedades estruturais da matriz.

Será considerada a multiplicação de matriz esparsa $\times$ vetor $y \leftarrow y + Ax$, onde A é uma matriz esparsa $m \times n$ com k elementos não-nulos, e x e y são vetores densos. A multiplicação matriz $\times$ vetor pode ser definida como

$$\forall a_{i,j} \neq 0 : y_i \leftarrow y_i + a_{i,j} \times x_j.$$

A implementação convencional da multiplicação matriz vetor está mostrada no Algoritmo 5.

Algorithm 5 Multiplicação de matriz densa-vetor.

```

for i = 0 to m - 1 do
  for j = 0 to n - 1 do
     $y[i] \leftarrow y[i] + v[i + m \cdot j] \cdot x[j]$ 
  end for
end for

```

No caso de A ser esparsa, um formato esparsa é o formato de coordenadas (COO), onde A é armazenada usando-se três vetores, v , ri e ci , todos com comprimento k . O elemento $v[l]$ armazena o valor do elemento da matriz na linha guardada em $ri[l]$ e na coluna salva em $ci[l]$. Usando este formato, a multiplicação matriz $\times$ vetor está mostrada no Algoritmo 6.

Algorithm 6 Multiplicação matriz $\times$ vetor em formato de coordenadas.

```

for  $l = 0$  to  $k - 1$  do
   $y[ri[l]] \leftarrow y[ri[l]] + v[l] \cdot x[ci[l]]$ 
end for

```

Outra classe de formatos é a que armazena faixas compactadas, que incluem os formatos CSR e CSC (*Compressed Sparse Column*) (LIN; CHUNG; LIU, 2003). CSR é uma coleção de vetores esparsos que permite acesso aleatório a todas as linhas e enumera eficientemente todos os elementos não-zeros de uma linha. O formato CSR está ilustrado na Fig. 20. A ideia é armazenar cada linha (mostradas com elementos da mesma cor) como um vetor esperso. Um único vetor de valores v armazena todos os valores do vetor esperso-linha de maneira ordenada, e um vetor de inteiros correspondente ci armazena todos os índices de colunas. Cada elemento p de um terceiro vetor armazena o deslocamento dentro de v e ci na linha i . O vetor p tem $m + 1$ elementos, onde o último elemento é igual ao número de elementos não-zeros. Esta estrutura de dados permite acesso aleatório a qualquer linha, e eficientemente enumera todos os elementos não-zeros de uma dada linha. Uma implementação da multiplicação matriz esparsa $\times$ vetor utilizando este formato está apresentada no Algoritmo 7.

Algorithm 7 Multiplicação matriz $\times$ vetor utilizando formato CSR.

```

for  $i = 0$  to  $m - 1$  do
  for  $l = p[i]$  to  $p[i+1] - 1$  do
     $y[i] \leftarrow y[i] + v[l] \cdot x[ci[l]]$ 
  end for
end for

```

Na multiplicação matriz $\times$ vetor, o paralelismo disponível através das linhas torna intuitivo e natural a escolha de distribuir o processamento correspondente a cada linha a um bloco de threads. O código mostrado no Algoritmo 8 aloca uma thread para efetuar os cálculos correspondentes a uma linha, e um bloco de threads para o tratamento de um conjunto de linhas. Isso corresponde a um mapeamento em que um grid unidimensional de blocos de threads e um bloco unidimensional de blocos de threads são usados para calcular a multiplicação matriz esparsa $\times$ vetor. Este código é muito similar ao apresentado no Algoritmo 7. Este formato é muito eficiente para armazenar a matriz esparsa, ocupando pouco espaço, e é fácil de implementar. Porém, não é otimizado para a necessidade de acesso coalescente e pode gerar diversas threads divergentes (VÁZQUEZ; FERNÁNDEZ; GARZÓN, 2011).

O formato ELLPACK-R, uma variante do ELLPACK original, é o atual padrão para implementação de operações de multiplicação matriz esparsa $\times$ vetor em GPGPUs (VÁZQUEZ; FERNÁNDEZ; GARZÓN, 2011). A ideia é comprimir as linhas ao deslocar todos os elementos não-zero para a esquerda, resultando em uma matriz $N \times N_{\max(nz)}$, onde N

Algorithm 8 Multiplicação matriz esparsa $\times$ vetor implementada em CUDA utilizando o formato CSR.

```

int line = blockDim.x * blockIdx.x + threadIdx.x ;
if line < dim then
    double sum = y[line];
    sum = 0;
    int start = p[line];
    int end = p[line + 1];
    for i = start to end - 1 do
        sum += x[ci[i]] * v[i];
    end for
    y[line] = sum;
end if

```

é o número de linhas e $N_{\max(nz)}$ é o maior número de não-zeros em uma linha.

Em contraste aos outros formatos de armazenamento de matrizes esparsas, no formato ELLPACK-R são, sim, armazenados valores nulos. Por exemplo, considere a matriz apresentada a seguir

$$A = \begin{pmatrix} 3 & 0 & 0 & 0 & 4 & 0 \\ 0 & 8 & 7 & 8 & 0 & 1 \\ 0 & 7 & 1 & 0 & 8 & 0 \\ 0 & 8 & 0 & 2 & 0 & 0 \\ 4 & 0 & 8 & 0 & 4 & 7 \\ 0 & 1 & 0 & 0 & 7 & 2 \end{pmatrix}$$

O formato ELLPACK-R para esta matriz fica

$$data = \begin{bmatrix} 3 & 4 & 0 & 0 \\ 8 & 7 & 8 & 1 \\ 7 & 1 & 8 & 0 \\ 8 & 2 & 0 & 0 \\ 4 & 8 & 4 & 7 \\ 1 & 7 & 2 & 0 \end{bmatrix} \quad idx = \begin{bmatrix} 0 & 4 & 0 & 0 \\ 1 & 2 & 3 & 5 \\ 1 & 2 & 4 & 0 \\ 1 & 3 & 0 & 0 \\ 0 & 2 & 4 & 5 \\ 1 & 4 & 5 & 0 \end{bmatrix} \quad rl = \begin{bmatrix} 2 \\ 4 \\ 3 \\ 2 \\ 4 \\ 3 \end{bmatrix}, \quad (3.1)$$

onde *data* contém a matriz comprimida (preenchida com zeros à direita), *idx* contém a coluna de cada elemento em *data* e *rl* contém o real comprimento de cada linha, antes do preenchimento com zeros - isto é, o número de não-zeros na linha. Como afirmado em (VÁZQUEZ; FERNÁNDEZ; GARZÓN, 2011), a incorporação de *rl* reduz a quantidade de “branches condicionais” no código em CUDA, aumentando a eficiência dos algoritmos.

Uma implementação simples da multiplicação matriz esparsa $\times$ vetor utilizando ELLPACK-R está ilustrada no Algoritmo 9. onde N é o tamanho da matriz, MNZ é o tamanho da linha com mais valores não-zeros. Lembrando que *data* e *idx* têm tamanhos $N \times MNZ$ e *rl* tem tamanho N .

Algorithm 9 Multiplicação matriz $\times$ vetor utilizando formato ELLPACK-R.

```

for i = 0 to N do
  for j = 0 to rl[i] do
     $y[i] \leftarrow y[i] + v[idx[i * MNZ + j]] \cdot data[i * MNZ + j]$ 
  end for
end for

```

O formato pJDS (padded Jagged Diagonals Storage) foi proposto como um aprimoramento ao formato ELLPACK-R original, com o objetivo de reduzir a quantidade de entradas nulas armazenadas desnecessárias, e melhorar a utilização dos recursos computacionais disponíveis (KREUTZER et al., 2012). No formato pJDS, as linhas do formato ELLPACK-R são ordenadas de acordo com seu tamanho. Em seguida, as linhas são divididas em blocos de acordo com o tamanho do warp, e todas as linhas em um bloco são preenchidas com zeros à direita de maneira que todas possuam o mesmo tamanho. Com isso, o armazenamento necessário é bastante reduzido. A matriz, à primeira vista, se torna menos uniforme, o que poderia interferir no acesso coalescente. Porém, o acesso à memória é mantido coalescente dentro do warp, em que todas as 32 threads estão executando a mesma instrução e acessando o mesmo bloco de memória, de forma sequencial. No exemplo da matriz A utilizando pJDS, a matriz será primeiro reordenada de acordo com o tamanho das linhas, como

$$ReorderedA = \begin{pmatrix} 8 & 0 & 7 & 1 & 0 & 8 \\ 0 & 4 & 8 & 7 & 4 & 0 \\ 7 & 8 & 1 & 0 & 0 & 0 \\ 1 & 7 & 0 & 2 & 0 & 0 \\ 0 & 4 & 0 & 0 & 3 & 0 \\ 8 & 0 & 0 & 0 & 0 & 2 \end{pmatrix}$$

e tem-se as seguintes estruturas, considerando warp hipotético de tamanho 2

$$data = \begin{bmatrix} 8 & 7 & 1 & 8 \\ 4 & 8 & 7 & 4 \\ 7 & 8 & 1 \\ 1 & 7 & 2 \\ 4 & 3 \\ 8 & 2 \end{bmatrix} \quad idx = \begin{bmatrix} 0 & 2 & 3 & 5 \\ 1 & 2 & 3 & 4 \\ 0 & 1 & 2 \\ 0 & 1 & 3 \\ 1 & 4 \\ 0 & 5 \end{bmatrix} \quad rl = \begin{bmatrix} 4 \\ 4 \\ 3 \\ 3 \\ 2 \\ 2 \end{bmatrix} \quad cs = \begin{bmatrix} 0 \\ 6 \\ 12 \\ 14 \\ 16 \end{bmatrix}, \quad (3.2)$$

onde $data$ contém a matriz reordenada comprimida (preenchida com zeros à direita, se necessário), idx contém a coluna de cada elemento em $data$, rl contém o real comprimento de cada linha, antes do preenchimento com zeros - isto é, o número de não-zeros na linha - e cs indica o início de cada linha, no vetor $data$. Este formato utiliza um vetor adicional, mas a necessidade de memória acaba sendo menor em casos em que as linhas não possuem todas o mesmo número de não-zeros. No exemplo dado, o formato ELLPACK-R precisou armazenar 54 valores, enquanto o formato pJDS armazenou 47 valores.

Uma implementação da multiplicação matriz esparsa $\times$ vetor utilizando pJDS está ilustrada no Algoritmo 10.

Algorithm 10 Multiplicação matriz $\times$ vetor utilizando formato pJDS.

```

for  $i = 0$  to  $N$  do
  for  $j = 0$  to  $rl[i]$  do
     $offset \leftarrow rs[j]$ 
     $y[i] \leftarrow y[i] + v[idx[offset + i]] \cdot data[offset + i]$ 
  end for
end for

```

Importante observar que todos estes algoritmos apresentados aqui utilizam matrizes “column-major”, isto é, com a coluna como principal coordenada.

A Fig. 21 ilustra a diferença entre o formato ELLPACK-R original e o formato pJDS. Neste exemplo, é considerada uma matriz de 12 linhas, e um warp de tamanho 2 (vide Fig. 21(a)). O formato de matriz ELLPACK-R está ilustrado na Fig. 21(b), enquanto o formato pJDS está na Fig. 21(c). Ambas as imagens mostram os tamanhos das linhas (em colunas) na cor escura. A cor clara indica o acesso e processamento desnecessários executados pelas threads por estarem em warps em que zeros foram acrescentados. Como pode ser visto, o formato pJDS ocupa menos memória para armazenar suas estruturas, realiza menos acessos desnecessários e, conseqüentemente, menos processamento vazio.

3.2 Paralelização de Solvers Triangulares através de Coloração de Grafo

Devido à natureza paralelizada das operações na GPU, é inevitável que ocorram tentativas de escrita concorrente. Por exemplo, se dois elementos compartilham um nó, as threads realizando cálculos para cada um desses elementos irão ambas precisar atualizar o mesmo dado. Para prevenir erros de sincronismo, técnicas que podem ser usadas incluem a utilização de operações atômicas, ou coloração utilizando árvore de dependência.

Considere o modelo MEF unidimensional simples com 5 nós, como mostrado na Fig. 22. Seus nós estão numerados sequencialmente, e a matriz de rigidez original é representada por uma matriz esparsa. A árvore de dependência mostra que nenhuma paralelização é permitida. Uma árvore de dependência representa os relacionamentos entre os elementos, e é definida da seguinte forma: uma aresta do nó i ao nó j significa que o valor relativo ao nó i influencia o valor referente ao nó j . Um método simples para verificar essa condição é verificar a linha j em uma matriz triangular L . Para cada coluna i com um elemento não-zero na linha j , uma aresta do vértice i ao vértice j é criada (KUMAR; KUMAR; BASU, 1993). O grau de dependência de cada nó na árvore de dependência por ser representado por cores. Nós que possuem o mesmo grau de dependência, isto é,

Figura 21 – Comparação do acesso à memória nos formatos ELLPACK-R and pJDS - o formato pJDS adiciona uma quantidade menor de “enchimento”. (a) Matriz exemplo, de tamanho 12×12 - a cor vermelha indica elementos não-nulos. (b) Matriz no formato ELLPACK-R: os elementos são compactados à esquerda, e zeros de enchimento são adicionados para que as linhas do mesmo warp possuam o mesmo comprimento (estes estão indicados na cor vermelho claro). (c) Matriz no formato pJDS: similar ao formato ELLPACK-R, exceto que as linhas são reordenadas de acordo com o tamanho antes da realização do enchimento com zeros. Warp hipotético de tamanho 2 threads.

(a)

(b) ELLPACK-R

(c) pJDS

Figura 22 – Modelo MEF com 5 nós numerados sequencialmente e sua matriz de rigidez. As cores indicam grau de dependência.

Figura 23 – Matriz triangular inferior obtida após a aplicação do pré-condicionador à matriz de rigidez do modelo MEF simples da Fig. 22.

possuem a mesma quantidade de nós abaixo dele na árvore de dependência, recebem a mesma cor.

A Fig. 23 apresenta a matriz triangular inferior obtida a partir aplicação do pré-condicionador à matriz de rigidez apresentada na Fig. 22. Como os nós foram numerados sequencialmente, a parte triangular inferior da matriz de rigidez não permite nenhuma forma de paralelização. Os elementos das matrizes representados por quadrados preenchidos com uma única cor possuem todos cores diferentes e, portanto, precisam ser processados sequencialmente na sua própria etapa de processamento. Os elementos da matriz representados por quadrados preenchidos com duas cores indicam dependências entre os elementos - e nós do modelo - e são calculados em etapas de propagação. A cor no canto esquerdo superior indica a etapa de processamento (cor) da qual depende, isto é, só pode ser processado após os elementos desta mesma cor terem sido processados. A cor no canto direito inferior indica a etapa em que será calculado - antes da etapa de processamento dos elementos desta cor, já que há dependência entre esses elementos (nós). Para esta configuração da malha, em particular, seriam necessárias 5 etapas de processamento e 4 etapas de propagação.

Se os nós fossem numerados de acordo com sua cor - obtida através da árvore de

Figura 24 – O mesmo MEF com 5 nós mostrado na Fig. 22, renumerado, e sua nova matriz de rigidez. Novamente, as cores indicam grau de dependência, com uma redução do número de cores necessárias.

Figura 25 – Matriz triangular inferior obtida após a aplicação do pré-condicionador à matriz de rigidez do modelo MEF ordenado por cor da Fig. 24.

dependências explicada na seção anterior, isto é, com os nós da mesma cor numerados sequencialmente, o modelo MEF se tornaria como mostrado na Fig. 24. Sua matriz triangular inferior obtida a partir da aplicação do pré-condicionador à matriz de rigidez está apresentada na Fig. 25. A substituição para frente – *forward substitution*, um procedimento utilizado no solver triangular – agora pode ser paralelizado, solucionando todos os nós da mesma cor simultaneamente. O resultado é que somente 2 etapas de processamento e 1 etapa de propagação são necessárias. Com isto, o solver triangular, um procedimento naturalmente sequencial, torna-se uma operação paralelizável.

Assim, temos que a malha do MEF pode ser abordada como um grafo, e deve ser colorida de tal forma que dois nós adjacentes nunca compartilhem a mesma cor. Tal algoritmo garante que nós adjacentes na malha terão cores distintas. Ao se reordenar os elementos, nós com a mesma cor são agrupados e sequencialmente ordenados, de forma que os valores, desconhecidos, associados a esses vértices podem ser calculados em paralelo no solver de sistemas triangulares (BERGER; BROUAYE; SYRE, 1982).

3.3 O formato pJDS Colorido para Armazenamento da Matriz

No caso modelo MEF simples apresentado na seção anterior, somente 2 cores foram necessárias. Pode ser mostrado que qualquer grafo plano pode ser colorido utilizando-se, no máximo, 5 cores. Esta propriedade pode ser utilizada para se obter maior paralelização e, portanto, o modelo - e, consequentemente, a matriz - precisa ser ordenado de acordo com a cor para que o solver possa ser paralelizado. Porém, o formato de matriz pJDS apresentado anteriormente precisa que a matriz esteja ordenada de acordo com o tamanho das linhas.

Para atender ambos os requisitos, é proposto neste trabalho um novo formato de armazenamento de matrizes esparsas baseado no pJDS, chamado Colored pJDS (CpJDS). Neste formato, a matriz deve ser ordenada duas vezes. Primeiro, é ordenada de acordo com a cor, dividindo suas linhas em blocos de cores. Em seguida, dentro de cada bloco de cor, as linhas são ordenadas de acordo com o tamanho. Por fim, as linhas são agrupadas em blocos do tamanho do warp, para o preenchimento dos zeros à direita, como estipulado pelo formato pJDS. É importante manter linhas de diferentes blocos de cor em warps distintos, o que exige que linhas vazias sejam adicionadas em algumas situações – a diagonal principal é preenchida com uns. A matriz resultante deve cumprir as seguintes regras:

1. Linhas são divididas em cores e ordenadas de acordo com a cor;
2. Linhas de cada cor são ordenadas de acordo com o tamanho e subdividas em blocos do tamanho do warp;
3. Todas as linhas do mesmo warp devem possuir o mesmo tamanho, então 0s são adicionados ao final da linha e 1, na diagonal principal;
4. A matriz pode ter linhas adicionadas de forma que todos os warps fiquem preenchidos;
5. Todos os elementos da diagonal principal são movidos para a primeira posição da linha.

A Fig. 26 ilustra este procedimento. A estrutura inicial da matriz é a mesma da matriz mostrada na Fig. 21, consistindo de 12 linhas, e a estrela indica o elemento da diagonal. Na Fig. 26(a), a matriz é colorida a partir da análise de dependências e, então, as linhas são compactadas à esquerda, restando apenas os elementos não-nulos, como mostrado na Fig. 26(b). Na Fig. 26(c), as linhas estão ordenadas de acordo com a cor, primariamente, e com o tamanho, secundariamente. Na Fig. 26(d), as cores claras com setas indicam elementos vazios que foram adicionados para se completar o tamanho do warp – inclusive novas linhas foram inseridas. O tamanho do warp hipotético considerado é de 2 threads. Além disso, os elementos da diagonal são enviados para a primeira posição da linha.

Isso é feito para que o acesso ao elemento da diagonal – importante para operações como o solver triangular e o pré-condicionador – seja realizado em bloco, de forma coalescente.

Um exemplo deste pré-processamento está ilustrado a seguir. Considere a seguinte matriz A , 5×5

$$A = \begin{pmatrix} 10 & 3 & 0 & 2 & 0 \\ 3 & 6 & 4 & 0 & 3 \\ 0 & 4 & 11 & 1 & 0 \\ 2 & 0 & 1 & 8 & 3 \\ 0 & 3 & 0 & 3 & 5 \end{pmatrix}.$$

A parte triangular inferior da decomposição de Cholesky incompleta é L , dada por

$$L = \begin{pmatrix} 3.16 & 0 & 0 & 0 & 0 \\ 0.95 & 2.26 & 0 & 0 & 0 \\ 0 & 1.77 & 2.80 & 0 & 0 \\ 0.63 & 0 & 0.52 & 2.69 & 0 \\ 0 & 1.33 & 0 & 1.41 & 0.74 \end{pmatrix}.$$

Um solver triangular utilizando L não pode ser facilmente paralelizado, e requer 5 etapas de cálculo de variável e 4 etapas de propagação. Porém, se A é colorida e ordenada de acordo com suas cores – a cor mais populosa primeiro, a matriz resultante pode ser paralelizada para

$$color_sorted_A = \begin{pmatrix} 10 & 0 & 0 & 2 & 3 \\ 0 & 11 & 0 & 1 & 4 \\ 0 & 0 & 5 & 3 & 3 \\ 2 & 1 & 3 & 8 & 0 \\ 3 & 4 & 3 & 0 & 6 \end{pmatrix}.$$

Agora, são necessárias apenas 2 etapas de processamento - solucionando 3 e 2 linhas simultaneamente, respectivamente - e apenas 1 etapa de propagação. A matriz transformada para o formato pJDS colorido fica

$$padded_color_sortedA = \begin{pmatrix} 10 & 0 & 0 & 0 & 2 & 3 \\ 0 & 11 & 0 & 0 & 1 & 4 \\ 0 & 0 & 5 & 0 & 3 & 3 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 2 & 1 & 3 & 0 & 8 & 0 \\ 3 & 4 & 3 & 0 & 0 & 6 \end{pmatrix}.$$

Figura 26 – Procedimento para criar o formato pJDS colorido. A estrela indica o elemento da diagonal. (a) Matriz da Fig. 21 colorida. (b) Matriz colorida com elementos não-nulos compactados à esquerda. (c) Matriz com as linhas reordenadas de acordo com a cor (primariamente) e tamanho (secundariamente). (d) Preenchimento com zeros – adicionados para que todas as linhas acessadas por threads no mesmo warp possam ter o mesmo tamanho, garantindo que todas as threads leiam blocos inteiros naturalmente alinhados, mostrados na cor mais clara, com setas indicando o crescimento da linha. Além disso, linhas também são adicionadas, para que todos os grupos de cor possam ter tamanho múltiplo do tamanho do warp (warp hipotético de 2 threads). Por fim, os elementos da diagonal são artificialmente colocados na primeira posição da linha, deslocando o resto da linha.

(a)

(b)

(c)

(d)

ou, na forma compactada,

$$padded_color_sortedA = \begin{pmatrix} 10 & 2 & 3 \\ 11 & 1 & 4 \\ 5 & 3 & 3 \\ 1 & 0 & 0 \\ 8 & 2 & 1 & 3 \\ 6 & 3 & 4 & 3 \end{pmatrix}.$$

Este formato aproveita vantagens provenientes dos formatos ELLPACK-R e pJDS, como acesso coalescente da memória, redução de “branches condicionais” e processamento homogêneo dentro de kernels. Também se aproveita da teoria de coloração de grafos para obter melhor paralelização – como todas as linhas dentro de uma cor são independentes entre si, o solver triangular pode passar por todas elas ao mesmo tempo.

A matriz é armazenada “column-major”, isto é, com os dados ordenados por coluna, em vetor de dados (“data”). Os índices de coluna são armazenados em um vetor de “índices”. O tamanho de cada linha é armazenado num vetor “rowSize”, e existe um vetor indicando o início de cada cor, “colors”. É necessário saber onde, no vetor de dados, começa cada coluna (vetor “columns”), além de saber onde está a primeira coluna de cada cor (vetor “colorsColumn”). Presente no algoritmo do solver triangular, também existe um vetor de dados para os valores do pré-condicionador, vetor “L”, que possui exatamente a mesma estrutura da matriz original. A matriz, simétrica, é sempre armazenada completa.

3.4 Operações Paralelizadas com Matriz e Vetor

Nesta seção, serão apresentados os algoritmos para implementar o produto interno de 2 vetores, a multiplicação matriz esparsa $\times$ vetor e o solver triangular.

3.4.1 Produto Interno de 2 Vetores

O produto interno de 2 vetores é uma operação altamente paralelizável. Por não envolver matriz, independente do formato da matriz apresentado, e sua implementação já foi bastante discutida, inclusive possuindo uma na própria biblioteca da nVidia. Sua paralelização, porém, é importante para o objetivo de paralelizar os algoritmos de Gradientes Conjugados, e será incorporada ao código desenvolvido neste trabalho por um motivo discutido mais à frente: a redução do número de funções e kernels invocados. O algoritmo serial está apresentado no Algoritmo 11.

A implementação paralela em CUDA precisa levar em consideração o tamanho dos vetores para sua paralelização. Considerando *Compute Capability* a partir de 2.0 – a mais atual é a 5.2, tem-se um limite de 1.024 threads por bloco de execução. Assim,

Algorithm 11 Produto interno de 2 vetores - algoritmo serial.

```

for idx = 0 to rows do
  resultado += v[idx] * u[idx]
end for
  
```

Figura 27 – Multiplicação ponto a ponto do produto interno. Cada thread realiza uma única multiplicação, e armazena o produto em um vetor de resultado parcial.

considerando vetores de tamanho N e blocos de tamanho 1.024 threads, tem-se que serão lançados $\text{ceil}(N/1024)$ blocos, onde ceil é a função *teto*, que arredonda para o próximo inteiro, se necessário. Além disso, cada thread realiza a multiplicação dos valores em índices iguais ao da thread, mas não é possível todas simultaneamente adicionarem ao resultado, eliminando todo o paralelismo. A forma padrão de solucionar isso é utilizar um vetor de cache em memória compartilhada, mais rápida que a memória global, e adicionar uma etapa de redução para totalizar a soma das multiplicações sempre às metades. A primeira etapa, da multiplicação ponto a ponto, está ilustrada na Fig. 27. Tem-se que cada thread realiza uma única multiplicação entre um valor no vetor v e um valor no vetor u , armazenando o resultado no vetor de resultados parciais p .

Em seguida, após a conclusão da etapa anterior, é preciso realizar a redução do vetor de resultados parciais. Na redução, metade das threads somam o valor no vetor de índice igual ao índice da thread com o valor na posição metade à frente, sucessivamente reduzido pela metade o número de elementos, como ilustrado na Fig. 28. Neste exemplo, considerou-se um vetor de tamanho 16. No caso da paralelização em CUDA, porém, o vetor parcial, que idealmente deve ficar na memória compartilhada, tem o escopo limitado às threads do bloco. Desta forma, não é possível realizar a redução em uma única etapa sem impor uma sincronização de threads em todos os blocos – o que não é possível em si – e sem o uso de memória global, para que cada bloco compartilhe o resultado da redução local. Desta forma uma única thread faria o trabalho de somar os resultados parciais de cada bloco.

Assim, utilizando CUDA C++, e com os vetores v , u , p – o resultado da redução local em cada bloco – e *resultado* (que não é um vetor, mas é referenciado como ponteiro)

Figura 28 – Redução do vetor de resultados parciais do produto interno. O número de threads ativas cai sempre pela metade. Neste exemplo, considerou-se a redução de um vetor de tamanho 16.

já alocados na memória global, pode-se invocar um kernel da seguinte forma:

```
inner_product<< blocks, 1024 >> (N, v, u, p)
```

Sendo `inner_product` o kernel definido a seguir

```
__global__ void inner_product(int size, float * v, float * u, float * p) {
    // shared memory for reduction
    __shared__ numType cache[BLOCKSIZE];

    // row index
    int row = blockDim.x * blockIdx.x + threadIdx.x;

    // cache index is the thread index inside a block
    int cacheIndex = threadIdx.x;
```

```

    if (row < size) {
        cache[cacheIndex] = v[row] * u[row];
    }
    else {
        cache[cacheIndex] = 0.0;
    }

    __syncthreads();

    int half = BLOCKSIZE >> 1;
    while (half != 0){
        if (cacheIndex < half) {
            cache[cacheIndex] += cache[cacheIndex + half];
        }

        __syncthreads();

        half >>= 1;
    }

    // totalizar o resultado parcial deste bloco
    if (cacheIndex == 0) {
        p[blockIdx.x] = cache[0];
    }

    // resta totalizar o resultado de todos os blocos
}

```

Para concluir a redução, bastaria chamar o kernel de redução

```
reduction<< 1, 1 >> (blocks, p, resultado)
```

Sendo `reduction` o kernel definido a seguir:

```

__global__ void reduction(int blocks, float * p, float * resultado) {
    float * sum = 0;
    if (for int i = 0; i < blocks; i++) {
        sum += p[i];
    }
    resultado[0] = sum;
}

```

Duas observações são importantes neste ponto. Primeiro, que o kernel `reduction` poderia, dependendo do tamanho, utilizar o mesmo algoritmo de redução tantas vezes quantas necessárias. Isso pode ser interessante quando o tamanho do vetor ficar muito grande. Segundo, é possível utilizar um método de sincronização baseado em escrita na memória global para sincronizar todas as threads, em todos os blocos – *threadfence*. Isso permite calcular o produto interno em um único kernel, sem a necessidade de um kernel para redução. O conceito da redução é o mesmo, porém.

Figura 29 – Iterações por coluna na multiplicação matriz esparsa $\times$ vetor. As colunas com cor escura indicam as leituras de memória – realizadas em blocos contíguos naturalmente alinhados. As threads ativas em cada iteração estão apresentadas à esquerda. São necessárias 7 iterações para multiplicar esta matriz de lado 14 por um vetor.

3.4.2 Multiplicação Matriz esparsa $\times$ Vetor

A multiplicação matriz esparsa $\times$ é uma operação totalmente paralelizável, uma vez que nenhuma linha depende da outra. No caso da implementação em CUDA, a utilização do formato de matriz apresentado apresenta vantagens na leitura da matriz, com todos os acessos em blocos consecutivos de memória, naturalmente alinhados, como ilustrado na Fig. 29. A mesma matriz foi utilizada para ilustrar o procedimento de montagem da matriz colorida (ver Fig. 26). Em cada imagem da Fig. 29, é possível observar as threads ativas – apresentadas à esquerda – e as posições de memória sendo lidas, em blocos consecutivos de tamanho que permite o alinhamento natural – de novo, é considerado, para ilustração, o hipotético warp de tamanho 2 e com bloco de leitura 8 bytes (2 leituras de 8 bytes), ao invés dos reais 32 threads para tamanho de warp e bloco de leitura de 128 bytes (32 leituras de 4 bytes).

O vetor de índices possui o mesmo tamanho e a mesma estrutura que o vetor de dados, e é acessado em blocos contíguos naturalmente alinhados. O vetor de tamanho de linhas tem tamanho igual ao número de linhas, e, portanto, cada thread acessa um valor no mesmo índice da thread, resultado em um acesso coalescido e eficiente. O acesso aos vetores relacionados a cor poderia ser mais problemático, por seus índices não possuírem qualquer relação com o número da linha. Porém, como estes vetores são pequenos – relacionados ao número de cores e às dependências, que podem ser mantidas pequenas, podem ser armazenados em memórias de constantes que, uma vez cacheadas, são muito rápidas – tão rápidas quanto ler do próprio registrador – contanto que todas as threads de um meio warp leiam o mesmo endereço de memória, o que é o caso pois todas as threads de um warp são da mesma cor. Por se tratar de uma matriz esparsa, o acesso ao vetor que está multiplicando a matriz é aleatório. Já o acesso às posições do vetor resultante é naturalmente alinhado e em blocos contíguos, já que os índices acessados são iguais aos das threads. O pseudo-código para a multiplicação matriz esparsa $\times$ vetor está ilustrado no Algoritmo 12.

Algorithm 12 Multiplicação matriz esparsa $\times$ vetor $A * x = y$, sendo A uma matriz no formato proposto, x e y vetores.

```

colorIdx = determineColor(row)
inColorIdx = row - colors[colorIdx]
firstColorColumn = colorsColumn[colorIdx]
sum = 0
for idx = 0 to rl[row] do
    columnStart = columns[firstColorColumn + idx]
    offset = columnStart + inColorIdx
    sum += data[offset] * v[indices[offset]]
end for
y[row] = sum

```

3.4.3 Solver Triangular

Um pseudo-código para o solver triangular com *forward substitution* utilizando pré-condicionador está apresentado no Algoritmo 13. Como algumas linhas são preenchidas com zeros, podem existir operações vazias. Porém, se todas as threads estiverem sincronizadas, uma operação vazia não afeta o tempo de processamento mais do que “branches condicionais” e acesso aleatório à memória, pois são realizadas em paralelo e não resultam em instruções adicionais.

O algoritmo para o solver triangular superior é semelhante ao do solver triangular inferior, com a diferença de que as cores são processadas inversamente, e está apresentado no Algoritmo 14.

Algorithm 13 Solver triangular inferior.

```

for k = 0 to colors - 1 do
  colorStart = colors[k]
  colorEnd = colors[k + 1]
  if row >= colorStart AND row < colorEnd then
    inColorIdx = row - colorStart
    firstColorColumn = colorsColumn[k]
    sum = 0
    for idx = 1 to rl[row] do
      columnStart = columns[firstColorColumn + idx]
      offset = columnStart + inColorIdx
      if indices[offset] < row then
        sum = sum + L[offset] * x[indices[offset]]
      end if
    end for
    x[row] = (b[row] - sum) / L[indices[columns[firstColorColumn] + inColorIdx]]
  end if
end for

```

Algorithm 14 Solver triangular superior.

```

for k = colors - 1 to 0 do
  colorStart = colors[k]
  colorEnd = colors[k + 1]
  if row >= colorStart AND row < colorEnd then
    inColorIdx = row - colorStart
    firstColorColumn = colorsColumn[k]
    sum = 0
    for idx = 1 to rl[row] do
      columnStart = columns[firstColorColumn + idx]
      offset = columnStart + inColorIdx
      if indices[offset] > row then
        sum = sum + L[offset] * x[indices[offset]]
      end if
    end for
    x[row] = (b[row] - sum) / L[indices[columns[firstColorColumn] + inColorIdx]]
  end if
end for

```

Figura 30 – Iterações por grupo de cor no solver triangular inferior e superior. Os blocos com cor escura indicam as leituras de memória – realizadas em blocos contíguos naturalmente alinhados. Cada linha é processada por uma thread. São necessárias 6 chamadas do kernel de solver triangular.

O processamento de cada cor em série, com todos os elementos de uma mesma cor processados simultaneamente, está ilustrado na Fig. 30. Na figura, pode-se observar que são necessárias $2 \times n_{cores}$ iterações para se resolver o solver completo – inferior seguido do superior. É importante acrescentar que a matriz armazenada para o cálculo do solver inclui ambas as matrizes triangulares resultantes da aplicação do pré-condicionador. Não somente isso, a diagonal principal – representada na figura por uma estrela – é movida para a “primeira coluna”, de forma que seu acesso é realizado em bloco contíguo, aumentando a eficiência. Por fim, a sequência de cores deve ser respeitada: ordem crescente de dependência para o solver triangular inferior, e ordem decrescente de dependência para o solver triangular superior.

3.5 Gradientes Conjugados Paralelizado

Algorithm 15 Gradientes Conjugados pré-condicionado.

```

1:  $r_0 = b - A * x_0$ 
2:  $z_0 = M^{-1} * r_0$ 
3:  $p = z$ 
4: for  $k = 0$  to  $N - 1$  do
5:    $\alpha_k = \frac{r_k^t \cdot z_k}{p_k^t * A * p_k}$ 
6:    $x_{k+1} = x_k + \alpha_k p_k$ 
7:    $r_{k+1} = r_k - \alpha_k A * p_k$ 
8:    $z_{k+1} = M^{-1} * r_{k+1}$ 
9:    $\beta_k = \frac{z_{k+1} \cdot r_{k+1}}{z_k \cdot r_k}$ 
10:   $p_{k+1} = z_{k+1} + \beta_k p_k$ 
11: end for

```

O algoritmo dos Gradientes Conjugados pré-condicionado está apresentado no Algoritmo 15. Cada iteração consiste de:

- 3× Produtos internos, (linhas 5 e 9)
- 3× Multiplicação escalar-vetor, (linhas 6, 7 e 10)
- 2× Somas de vetor, (linhas 6 e 10)
- 1× Solvers triangulares (inferior e superior), (linha 8)
- 1× Subtração de vetor, (linha 7)
- 1× Multiplicação de matriz esparsa-vetor. (linha 5)

Todas essas operações podem ser paralelizadas de forma que o tempo de processamento não varie com o tamanho da malha utilizada – até o ponto de saturação da placa de vídeo, pelo menos. Todos os kernels possuem uma thread manipulando uma linha da matriz (ou vetor). O solver triangular é o único kernel que necessita de um laço adicional – no número de cores. Portanto, independentemente do tamanho da matriz, o número de passos necessários não varia – desde que os kernels sejam invocados individualmente para cada cor e até o ponto de saturação da placa de vídeo, diferentemente das implementações seriais.

Assim, pode-se ter o pseudo-código para a implementação dos Gradientes Conjugados utilizando GPU mostrado no Algoritmo 16. u , e q são vetores intermediários auxiliares.

Algorithm 16 Gradientes Conjugados pré-condicionado utilizando kernels para paralelização em GPU usando CUDA.

```

mult_matriz_vetor( $A, x, u$ )// $u = A * x$ 
vetor_subtr( $b, u, x$ )// $x = b - u$ 
for  $k = 0$  to  $colorCount$  do
     $colorStart = colors[k]$ 
     $colorEnd = colors[k + 1]$ 
    solver_triangular_inferior( $L, r, u, colorStart, colorEnd$ )// $u = L^{-1} * r$ 
end for
for  $k = colorCount$  to  $0$  do
     $colorStart = colors[k]$ 
     $colorEnd = colors[k + 1]$ 
    solver_triangular_superior( $U, u, z, colorStart, colorEnd$ )// $z = U^{-1} * u$ 
end for
 $p = z$ 
for  $k = 0$  to  $N$  do
    inner_product( $r, z, rz$ )
    mult_matriz_vetor( $A, p, q$ )// $q = A * p$ 
    inner_product( $p, q, pq$ )
     $alpha = rz/pq$ 
    vetor_escalar( $p, alpha, u$ )// $u = alpha * p$ 
    vetor_soma( $x, u, x$ )// $x = x + u$ 
    vetor_escalar( $q, alpha, u$ )// $u = alpha * q = alpha * A * p$ 
    vetor_subtr( $r, u, r$ )// $r = r - u$ 
    for  $k = 0$  to  $colorCount$  do
         $colorStart = colors[k]$ 
         $colorEnd = colors[k + 1]$ 
        solver_triangular_inferior( $L, r, u, colorStart, colorEnd$ )// $u = L^{-1} * r$ 
    end for
    for  $k = colorCount$  to  $0$  do
         $colorStart = colors[k]$ 
         $colorEnd = colors[k + 1]$ 
        solver_triangular_superior( $L^t, u, z, colorStart, colorEnd$ )// $z = (L^t)^{-1} * u$ 
    end for
     $rz_{old} = rz$ 
    inner_product( $r, z, rz$ )
     $beta = rz/rz_{old}$ 
    vetor_escalar( $p, beta, u$ )// $u = beta * p$ 
    vetor_soma( $z, u, p$ )// $p = z + u$ 
end for

```

3.6 Decomposição de Cholesky Incompleta

O algoritmo apresentado anteriormente exige que o pré-condicionamento seja aplicado para decompor a matriz A em duas matrizes triangulares, inferior e superior, tais que $A = L * L^t$, onde A é a matriz completa original, simétrica e positiva definida, e L é a matriz triangular inferior e L^t , a sua transposta, é a matriz triangular superior. No caso de A esparsa, porém, é interessante o uso da decomposição de Cholesky incompleta, por manter a esparsidade da matriz original, ao contrário da decomposição completa, que torna a matriz A “cheia”.

A implementação simples (apresentada em Algoritmo 4) é, por natureza, serial, pois cada coluna subsequente depende da anterior. A Fig. 31(a) ilustra a progressão, no triângulo amarelo, do elemento azul na diagonal. O elemento da diagonal é calculado; este valor é propagado para o restante da sua coluna (em azul claro); por fim, os valores da coluna azul claro são propagados para a submatriz triangular inferior indicada pelo triângulo amarelo (diagonal, em vermelho, inclusa). Porém, considere uma matriz reagrupada em 4 cores ilustrada na Fig. 31. Utilizando o algoritmo original (ver Algoritmo 3), tem-se que os elementos da diagonal dependem apenas dos elementos anteriores de sua linha. Os elementos não-diagonais dependem dos elementos anteriores de sua linha e também dos elementos anteriores na linha relativa a sua diagonal, e do próprio elemento da diagonal (ver Fig. 31(b)). Graças ao esquema de coloração, todos os nós de uma mesma cor não possuem dependência entre si. Isso significa que, debaixo da diagonal, há uma submatriz triangular onde todos os elementos são nulos. Portanto, todos os elementos da diagonal de uma cor, podem ser calculados simultaneamente e, em seguida, todos os elementos abaixo das diagonais desta cor podem ser calculados simultaneamente, como mostrado na Fig. 31(b). Como resultado, a decomposição de Cholesky incompleta pode ser calculada em um número de iterações igual ao número de cores, sendo cada iteração composta do cálculo dos elementos da diagonal, seguido do cálculo dos elementos abaixo.

3.7 Resultados

Para a avaliação da paralelização utilizando o formato pJDS colorido, os kernels desenvolvidos foram testados em um computador equipado com a placa GPU NVIDIA GeForce GTX980, com 2.048 núcleos CUDA (do inglês, *CUDA cores*), 4GB DDR5 de memória DRAM, utilizando Compute Capability 5.2 e CUDA 7.5. A CPU é um Intel Xeon ES645 2,4GHz com 32GB RAM, rodando Windows 7 Professional. Todos os kernels utilizam precisão simples para dados em ponto flutuante, e são lançados em blocos de 512 threads para malhas pequenas – até cerca de 2.500 nós – e 1.024 threads para as demais. Os algoritmos apresentados foram comparados a uma implementação serial, Eigen2. Todos os tempos foram medidos considerando cinco diferentes malhas de tamanhos similares,

Figura 31 – Decomposição incompleta de Cholesky (a) progressiva e serial; (b) em paralelo, devido à coloração.

e os tamanhos mostrados representam as médias de tais malhas. Para a coloração, foi utilizado um algoritmo de *backtracking* simples, e as cores estarão indicadas. Este não é o melhor algoritmo para coloração, sendo capaz de dividir as malhas em 5 ou 6 cores. O pré-condicionador utilizado foi a decomposição de Cholesky incompleta, que também foi avaliado.

3.7.1 Medindo Tempo de Execução

Para se avaliar os tempos, foram utilizadas duas abordagens principais, utilizando medição de tempo em CPU e em através de eventos na GPU. Para realizar a medição em CPU, como as chamadas de kernel são eventos assíncronos – isto é, uma vez invocado o kernel, o programa em CPU continua sem esperar pelo término do processamento –, utilizou-se da diretiva *cudaDeviceSynchronize* para bloquear a CPU até que todo os processamentos fossem concluídos. A outra maneira, utiliza eventos disparados pela própria GPU, alertando os pontos de início e fim de processamento da fila. Para muitas repetições – a partir de 1.000 repetições – a diferença na medição fica abaixo de 0,1%. Cada avaliação tem o número de repetições apresentado, mas sempre utilizou-se, pelo menos, 1.000 repetições. Assim, os tempos apresentados podem utilizar uma forma ou outra, sem a distinção de qual forma de medição foi utilizada.

As medidas obtidas com os métodos mencionados, porém, envolvem diversas variáveis como tempo de invocação da API, tempo de enviar instrução para a placa de vídeo pelo driver, latência de início de execução de kernel e tempo de sincronização, além do tempo da execução do kernel em si. Uma ferramenta utilizada para uma melhor avaliação de desempenho foi a Nsight Visual Studio, que possui, entre outras funcionalidades, uma ferramenta bastante completo. Com ela, é possível verificar as filas de execução da API, do driver da placa de vídeo, dos kernels, obter medidas de desempenho e traçar uma linha

Figura 32 – Linha do tempo de execução de um kernel vazio.

do tempo de execução.

Para estimar os tempos que não são de interesse, foi criado um kernel vazio, que é invocado diversas vezes. Este kernel não recebe nenhum parâmetro e, portanto, sua sobrecarga de chamada é o menor possível.

Uma chamada de kernel é traduzida pela API para algumas funções básicas que configuram a chamada e realizam a invocação em si. No caso de um kernel que não recebe nenhum parâmetro, tem-se que a invocação do kernel resulta em duas chamadas da API: `cudaConfigureCall` e `cudaLaunch`, sendo que esta última é a que emite a instrução de execução do kernel. As instruções são colocadas nas filas do driver – uma de espera e uma de enviada para execução – e só então são, de fato, executadas. O resultado disso é um latência para iniciar a execução do kernel que pode ser vista na Fig. 32. É possível visualizar um trecho da linha do tempo da chamada de um kernel vazio repetidas vezes, destacando-se a primeira chamada. A linha superior é a linha de chamada em CPU, já traduzida para chamadas da API. A execução da API CUDA está na segunda linha. As demais linhas representam, neste caso com apenas um kernel vazio, a execução deste kernel. A latência entre a invocação do kernel em CPU e o início efetivo deste foi medida através do profiler, resultando em aproximadamente $140 \mu s$, um valor muito maior do que o tempo de execução do kernel em si - em média, $1,3 \mu s$ por execução. É importante observar que a utilização do profiler Nsight com a captura dessas informações adiciona uma latência maior do que a real, mas mesmo com essa latência adicional, é possível observar a ordem de grandeza.

A latência da primeira execução, porém, é diluída entre todas as execuções. Existe, também, a latência para a sincronização CPU-GPU usando `cudaDeviceSynchronize`, para a medição dos tempos. Dito isso, o tempo total médio para a execução de 1.000.000 de repetições foi de $9,3 \mu s$ para a chamada de um kernel vazio. Isso nos permite estimar em $8 \mu s$ como a latência média de execução de um kernel. Isso também significa que, no caso de um kernel extremamente rápido, o gargalo de execução é a API CUDA. Isso pode ser observado na linha do tempo da Fig. 33. Os espaços entre os blocos de execução – que

Figura 33 – Linha do tempo de execução de um kernel vazio – espaçamento entre blocos de execução indicam GPU ociosa.

Tabela 1 – Tempo total médio de execução (em ms) para o produto interno - 1.000 repetições.

Tamanho	Eigen2	CUDA - GPU	CUDA - CPU
1.046	1,2	4,4	9,3
4.450	4,2	4,0	9,3
9.500	8,8	4,2	9,2

podem ser vistos na terceira linha de blocos – de kernel indicam que a GPU está ociosa, aguardando instrução para executar.

Assim, todos os tempos medidos, para as operações com matrizes esparsas e vetor paralelizadas, devem apresentar o tempo em que o kernel está, de fato, em execução e o tempo real imposto no algoritmo que o utiliza – ou seja, considerando as latências de chamada.

3.7.2 Avaliando os Kernels Individualmente

A Tabela 1 mostra uma comparação entre o Eigen2 – utilizando CSR – e a implementação em CUDA para o produto interno de dois vetores. Os tamanhos dos vetores estão indicados, e as operações foram repetidas 1.000 vezes, com este experimento repetido 10 vezes. Há duas colunas mostrando o tempo de execução em GPU, sendo uma o tempo real de execução do kernel e a segunda coluna indicando o tempo médio entre a chamada do kernel e a conclusão. Como pode ser visto, uma implementação serial como o Eigen é significativamente afetada pelo tamanho dos vetores, diferentemente da implementação em paralelo em GPU. O kernel do produto interno paralelizado levou aproximadamente 4ms para todos os tamanhos de vetores considerados, o que é um resultado pior que a versão serial para pequenos vetores, mas se torna gradativamente mais rápido, comparativamente, conforme os vetores crescem. Observando o tempo total de execução para a CPU, é interessante notar que o tempo é o mesmo para o kernel vazio, mesmo o kernel ficando mais de 3 vezes mais tempo em execução. Isto reforça a ideia de que a GPU está subaproveitada por conta da latência da API.

Tabela 2 – Tempo total médio de execução (em ms) para a multiplicação matriz esparsa $\times$ vetor – 1.000 repetições.

Tamanho	Eigen2	CUDA - GPU	CUDA - CPU
1.046	44,4	7,7	14,4
4.450	192,3	7,9	24,2
9.500	338,1	11,3	37,6

Figura 34 – Linha do tempo de execução para o produto matriz esparsa $\times$ vetor. Espaçamento entre blocos de execução indicam GPU ociosa, desta vez o gargalo está na CPU.

A Tabela 2 mostra uma comparação entre o Eigen2 e a implementação em CUDA para a multiplicação matriz esparsa $\times$ vetor. Os tamanhos das matrizes e vetores estão indicados, e as operações foram repetidas 1.000 vezes, com este experimento repetido 10 vezes. Há duas colunas mostrando o tempo de execução em GPU, sendo a primeira o tempo real de execução do kernel e a segunda coluna indicando o tempo médio entre a chamada do kernel e a conclusão. De novo, a implementação serial do Eigen é significativamente afetada pelo tamanho das matrizes, diferentemente da implementação em paralelo em GPU. O kernel da multiplicação matriz esparsa $\times$ vetor paralelizado foi efetivamente mais rápido que a implementação serial para todos os casos. Porém, novamente é interessante notar a diferença entre os tempos de kernel e para a CPU.

A Fig. 34 apresenta um trecho da linha do tempo para a multiplicação matriz esparsa $\times$ vetor, para a malha de 9.500 nós. É possível observar que a GPU está ociosa, novamente, mas desta vez está aguardando, indiretamente, que a CPU envie acionamentos da API. Isso significa que o código de acionamento precisa ser otimizado, ainda que, para a geração deste teste, a CPU não estivesse fazendo nenhuma operação intensiva.

Por fim, a Fig. 35 apresenta o resultado dos testes para um conjunto muito maior de matrizes, e uma linha de tendência é apresentada. Como esperado, existe muita vantagem em utilizar a GPU para a multiplicação matriz esparsa $\times$ vetor.

Para a validação da estrutura de matriz apresentada neste trabalho, a Tabela 3

Figura 35 – Comparação do tempo em relação ao número de nós para a multiplicação matriz esparsa $\times$ vetor, Eigen vs. CUDA com o formato de matriz proposto. Tempo em relação ao número de nós.

Tabela 3 – Tempo total médio de execução (em ms) para a multiplicação matriz esparsa $\times$ vetor – 1.000 repetições.

Tamanho	CjPDS	CSR	ELLPACK
1.046	7,74	5,89	6,96
2.590	7,37	6,48	7,86
4.450	7,89	8,15	10,02
7.850	8,09	12,40	15,70
9.500	11,27	14,72	17,14
15.600	12,25	23,13	27,90

Tabela 4 – Tempo total médio de execução (em ms) para o solver triangular - 1.000 repetições.

Tamanho	Cores	Eigen2	CUDA - GPU	CUDA - CPU
1.046	5	28,5	78,5	134,7
4.450	6	113,1	95,6	160,3
9.500	6	217,0	111,6	181,1

compara o desempenho do kernel desse formato com os outros comentados, CSR e ELLPACK. É possível observar que o controle adicional, através da coloração, possui uma sobrecarga inicial que afeta o desempenho em matrizes pequenas. Conforme o tamanho da matriz aumenta, porém, rapidamente o formato proposto passa a levar vantagem.

A Tabela 4 mostra uma comparação entre o Eigen2 e a implementação em CUDA usando pJDS colorido o solver triangular inferior seguido do superior. Os tamanhos dos

Figura 36 – Linha do tempo de execução para o solver triangular – inferior e superior. Espaçamento entre blocos de execução indicam GPU ociosa.

Figura 37 – Comparação do tempo em relação ao número de nós para o solver triangular completo – inferior e superior, Eigen vs. CUDA com o formato de matriz proposto.

vetores e matrizes e o número de cores utilizados estão indicados, e as operações foram repetidas 1.000 vezes. No caso do solver triangular, é utilizado um laço adicional em CPU, iterando pelas cores. No caso deste kernel, a implementação serial é mais rápida para as menores malhas, mas o que chama a atenção é o fato de o tempo efetivo para a CPU ser o dobro do tempo de execução dos kernels. Um dos motivos para isso pode ser visto na Fig. 36: o fato de serem necessários invocar tantos kernels adiciona muita latência para o início de execução. O desempenho do solver triangular para malhas diversas pode ser visto na Fig. 37.

Observa-se que a comparação do formato proposto com os demais formatos pré-existentes para o solver triangular não foi efetuado pelo fato de os outros formatos não poderem ser paralelizados, resultando em operações sequenciais. A vantagem do formato apresentado neste trabalho é evidente.

Figura 38 – Diagrama de sequência do algoritmo do métodos dos Gradientes Conjugados - inicialização.

3.8 Consolidação de Kernels

Os maiores problemas dos kernels avaliados até o momento não estão diretamente relacionados aos kernels, em si, mas ao restante da arquitetura utilizada. Seja por causa das latências da API CUDA ou driver, seja porque a CPU não está conseguindo enviar instruções suficientes para manter a GPU em funcionamento pleno. O uso de paralelismo dinâmico – com kernels invocando outros kernels, que aparentemente eliminaria a variável da CPU, rendeu resultados equivalentes. Assim, a solução proposta é a redução da modularização e junção dos kernels.

Considere novamente o algoritmo dos Gradientes Conjugados, apresentado no Algoritmo 16. Ele pode ser ilustrado com um diagrama de sequência, apresentado nas Fig. 38 e Fig. 42. Na Fig. 38, tem-se a inicialização das variáveis. São chamados 4 kernels e uma cópia de memória.

O primeiro kernel chamado efetua uma multiplicação matriz esparsa $\times$ vetor, utilizando todas as threads invocadas, uma thread por linha. Em seguida, é chamado um kernel de subtração de vetores, também com uma thread por linha e utilização completa das threads. O terceiro kernel, solver triangular inferior, é chamado uma vez por cor, podendo ser invocado com o número de threads igual ao tamanho da cor ativa. Isso significa que a utilização da GPU, em termos de quantidade de threads, é inferior se comparado com outros kernels. O mesmo ocorre para o solver superior.

Figura 39 – Diagrama de sequência de execução do solver triangular. O kernel é invocado uma vez para cada cor. A quantidade de threads pode ser igual ao número de elementos da cor. As setas significam threads.

Unificar os dois primeiros kernels é trivial, uma vez que as threads têm sua atuação restrita para uma única linha. Os solvers triangulares atualmente possuem o comportamento mostrado na Fig. 39. A quantidade de threads utilizadas pode ser igual ao tamanho da cor ou ao tamanho da matriz – neste último caso, existe um desperdício de threads. Na Fig. 39, os blocos de cores indicam as threads ativas, processando o resultado para aquela cor, no solver triangular – as demais threads estão ociosas. É apresentado o comportamento com a quantidade de threads igual ao tamanho da matriz pois, para fazer a junção dos kernels, o kernel resultante precisará ser invocado com uma thread por linha para que o comportamento das demais operações se mantenha.

Se as chamadas ao kernel de solver triangular forem unificadas em uma única

Figura 40 – Diagrama de sequência de execução do solver triangular ilustrando o problema da falta de sincronização entre as cores. O kernel é invocado uma única vez e a iteração por cor foi migrada para dentro do kernel. A quantidade de threads utilizadas é igual ao tamanho da matriz. As setas significam threads.

invocação de kernel, com a iteração de cor passada para dentro dele, surgirá um problema de sincronização, ilustrado na Fig. 40: as threads que não pertencem ao grupo de cor ativo passariam imediatamente para a próxima cor, o que faz com que as cores não sejam processadas em série, levando a resultados incorretos. Diferentemente do produto interno, no caso do solver triangular não é realizada uma operação de escrita à memória global pelos blocos de threads ociosos, impossibilitando o uso de *threadfence*. Poderia-se usar um vetor *dummy* para tentar forçar essa sincronização, mas isso significaria poluir instruções de memória, diminuindo o throughput dos acessos pelas threads que estão, de fato, realizando operação.

Uma opção que resolveria essa situação está apresentada na Fig. 41. Pode-se limitar o processamento do solver triangular para um único bloco de threads e, caso o tamanho da cor seja maior do que o tamanho do bloco, realizar o janelamento adequado. Isto implica em abandonar as threads ociosas, para que elas encerrem sua execução imediatamente. Considerando um tamanho de bloco de 1.024 threads, a operação de solver triangular ainda estaria altamente paralelizada e utilizando, no caso da placa GeForce GTX980, metade de seus núcleos CUDA. Isto não é realmente um problema, já que essa capacidade de processamento não utilizada pode ser direcionada para outras operações. Por fim, limitar o processamento do solver triangular para um único bloco também adiciona um elemento de variação linear com o tamanho das matrizes.

Figura 41 – Diagrama de sequência de execução do solver triangular, utilizando um único bloco. O kernel é invocado uma única vez e a iteração por cor foi migrada para dentro do kernel. A quantidade de threads utilizadas é igual ao tamanho da matriz. As setas significam threads.

Um ponto importante a se levar em consideração com esta abordagem é a impossibilidade de realizar outras operações que utilizam uma thread por linha após o solver triangular. Assim, o kernel deve ser encerrado após o solver triangular.

Para a inicialização, restou realizar apenas a cópia de memória. Porém, observando o diagrama de sequência de kernels de cada iteração dos Gradientes Conjugados, apresentado na Fig. 42, tem-se que o primeiro kernel de iteração precisa terminar com o solver triangular, para que a operação seguinte possa iniciar com todas as threads ativas. O segundo kernel, com um produto interno, redução, multiplicação escalar-vetor, e soma de vetores pode ser realizado, inclusive, no início da iteração. Afinal o produto interno entre r e z realizado logo no início da iteração é exatamente o mesmo realizado logo após o solver triangular da iteração anterior.

O diagrama de sequência dos Gradientes Conjugados reordenado está ilustrado na Fig. 43. Basta iniciar $rk_0 = 1$. Desta forma, é possível compactar todos os kernels em um único kernel.

Um outro ponto interessante é a discussão do uso de *threadfence* para sincronização de threads de blocos diferentes no produto interno, uma estratégia muito utilizada. Utilizar *threadfence* significaria que todos os kernels de uma iteração dos gradientes conjugados

Figura 42 – Diagrama de sequência do algoritmo dos Gradientes Conjugados - iteração.

Figura 43 – Diagrama de sequência reordenado do algoritmo dos Gradientes Conjugados - iteração.

Tabela 5 – Comparação dos algoritmos dos Gradientes Conjugados Pré-condicionados - tempo médio (em ms) para 30 iterações

Tamanho	Cores	Eigen2	CUDA - GPU	CUDA - CPU
1.046	5	3,1	2,6	3,0
2.590	6	7,2	3,6	4,0
4.450	6	11,9	4,2	5,0
6.900	6	19,5	5,8	6,0
9.500	6	26,9	7,7	7,8

reordenado poderiam ser consolidados em um único kernel, algo bastante desejável por diminuir a latência de chamada de kernels para o valor mínimo. Esta estratégia, porém, pode não ser tão interessante no caso do problema inverso, porém. O motivo disso é a utilização de streams para invocação de kernels distintos, relativos aos diferentes problemas diretos sendo processados em paralelo. O uso de *threadfence* implicaria utilizar eventos na memória global, podendo impactar o desempenho de kernels rodando em outros streams. Neste caso, o uso de 3 kernels seria o mais recomendado, pois como é demonstrado no capítulo seguinte, a latência de chamada de kernels adicional é diluída pela maior saturação da placa de vídeo obtida com o uso de streams. Assim, a divisão de kernels ocorreria após os produtos internos e após o solver triangular.

3.9 Resultados para Kernels Consolidados

A Tabela 5 mostra uma comparação entre o Eigen2 e a implementação em CUDA usando pJDS colorido o algoritmo dos Gradientes Conjugados pré-condicionado, para 30 iterações. Os tamanhos dos vetores e matrizes e o número de cores utilizados estão indicados. A vantagem da paralelização apresentada está presente mesmo na menor malha utilizada. Além disso, é possível observar que os tempos efetivos para a CPU estão muito mais próximos dos tempos dos kernels, o que indica uma melhor utilização da GPU, como pode ser visto na Fig. 44. Na Fig. 45, pode-se ver a comparação entre os métodos serial e paralelo, com a indicação da tendência. Utilizar o método paralelo tornou-se muito vantajoso, como mostrado da Fig. 46, e a escolha de um algoritmo de coloração mais eficiente poderia aumentar ainda mais os ganhos apresentados.

O resultado da utilização dos Gradientes Conjugados paralelizado proposto na malha de 2.590 nós está ilustrado na Fig. 47, com a execução de todas as iterações. Sendo o resultado do problema direto, a imagem representa a distribuição de potencial elétrico para uma distribuição de condutividades conhecida – foi utilizada uma simulação de cuba vazia, preenchida com solução salina – e corrente elétrica conhecida, aplicada no padrão “pula três”.

A Tabela 6 apresenta os resultados da paralelização da decomposição de Cholesky

Figura 44 – Linha do tempo de execução para o algoritmo de gradientes conjugados pré-condicionado utilizando consolidação em 3 kernels.

Figura 45 – Comparação do tempo em relação ao número de nós para o algoritmo dos Gradientes Conjugados pré-condicionado, Eigen vs. CUDA com o formato de matriz proposto. Média para 30 iterações.

Tabela 6 – Comparação dos algoritmos de decomposição de Cholesky incompleta – tempo em ms

Tamanho da malha (nós)	Eigen2	CUDA
1.046	0,13	4,75
4.450	0,52	9,44
9.500	1,12	14,98

incompleta. É possível perceber que a implementação paralela é extremamente penalizada pela grande quantidade de acessos aleatórios à memória. Com isso, a melhor abordagem é calcular o pré-condicionador na CPU e copiar o vetor de dados para GPU. No caso do problema direto, isso é feito em sequência. A Fig. 48 mostra o aumento linear do tempo de execução para a implementação serial, contra um aumento mais acentuado para a implementação paralela.

Figura 46 – Comparação do ganho de velocidade no problema direto paralelizado em GPU proposto em relação à implementação serial usando Eigen.

Figura 47 – Distribuição de potencial elétrico – resultado do problema direto paralelizado para a malha de 2.590 nós e padrão de corrente “pula três”.

Figura 48 – Comparação do tempo em relação ao número de nós para a decomposição de Cholesky incompleta, Eigen vs. CUDA com o formato de matriz proposto.

Tabela 7 – Pré-processamento - tempo médio para 100 iterações

Tamanho da malha (nós)	CUDA (ms)
1.046	25
2.130	138
4.500	708
6.900	1.748
9.050	3.152

Por fim, é importante observar que os algoritmos paralelizados em GPU utilizando o formato pJDS colorido exigem que a malha seja pré-processada. Este tempo - apresentado na Tab. 7 - é fixo e pode ser descartado do tempo total de processamento, já que só precisa ser executado uma única vez para cada malha nova - a estrutura pode ser salva e carregada para uso posterior. Como esperado, esse pré-processamento depende do tamanho da malha.

3.10 Conclusão

Neste capítulo, foi estudada a programação massivamente paralela em GPU, e foi proposta uma nova representação de matriz esparsa, a pJDS colorida, que foi efetivamente mais rápida que a implementação serial do Eigen2 para todos os kernels especificamente desenvolvidos: produto matriz $\times$ vetor e solver triangular. O novo formato proposto também mostrou-se mais eficiente para matrizes maiores em relação a formatos muito utilizados.

Foi identificado que o acionamento de diversos kernels em sequência não foi efetivo,

deixando a GPU utilizada ociosa por longos períodos. Assim, para evitar a ociosidade, os kernels foram agrupados, com exceção do solver triangular, pois precisa fazer a sincronização de threads não utilizadas, e do produto interno entre dois vetores, que foi separado em duas partes.

Assim, o método direto foi solucionado de modo massivo em GPU.

4 Reconstrução de imagens de TIE através do Problema Inverso

Como existem métodos conhecidos para solucionar eficientemente o problema direto (como o MEF), o problema inverso pode ser compreendido como um problema de otimização, onde as variáveis de otimização são os parâmetros de condutividade internos ao domínio (por exemplo, o domínio é dividido em segmentos contínuos e a condutividade interna de cada segmento — presumidamente constante — se torna um parâmetro) e a função de otimização é uma medida de como a solução do problema direto aplicada à distribuição de condutividade produzida pelas variáveis de otimização se aproximam dos dados medidos.

Uma função objetivo possível E é a distância Euclidiana entre os potenciais elétricos medidos ϕ_m^i e os potenciais calculados ϕ_c^i nos eletrodos para todos os padrões de corrente aplicados

$$E = \sqrt{\sum |\phi_m^i - \phi_c^i|^2}. \quad (4.1)$$

Um exemplo desta proposta foi apresentado por Mello et al. (MELLO et al., 2008), onde a função objetivo (4.1) é minimizada por programação linear sequencial objetivando determinar a distribuição de condutividades. Mello et al. (MELLO et al., 2008) afirmaram que a solução para este problema de otimização por métodos que requerem a determinação do gradiente da função objetivo é muito difícil pois os erros associados a estes cálculos podem ser tornar demasiadamente grandes devido ao fato deste ser um problema mal condicionado.

Esta é a principal motivação na utilização do recozimento simulado para solucionar a TIE, pois ele não requer a avaliação de derivadas da função objetivo.

4.1 Recozimento Simulado

O recozimento simulado (KIRKPATRICK; GELLAT; VECCHI, 1983) é a meta-heurística probabilística empregada neste trabalho, e é considerado uma meta-heurística probabilística de exploração local. Este tipo de heurística de otimização analisa a cada iteração uma e somente uma solução do problema. O custo desta solução é comparado com o custo da solução da iteração imediatamente anterior, e baseando-se em regras da heurística, uma nova solução é gerada para a próxima iteração.

No caso do recozimento simulado, a solução do problema x_i^* avaliada na iteração $i+1$ é escolhida aleatoriamente em uma vizinhança da solução atual $V(x_i)$. Por vizinhança de uma solução, entende-se um conjunto de soluções válidas relacionadas a uma determinada solução. A definição de vizinhança é diretamente dependente do problema de otimização abordado, e da maneira específica com a qual se decide aplicar o recozimento simulado ao problema. Cabe notar que a distribuição da probabilidade de seleção das soluções dentro da vizinhança não é necessariamente uniforme.

A solução atual x_i pode ser substituída em uma determinada iteração pela solução analisada x_i^* de acordo com as seguintes regras:

- $f(x_i^*) \leq f(x_i)$: a solução analisada na iteração tem custo inferior ou igual ao da solução atual. Neste caso, a solução atual é imediatamente substituída pela solução analisada.
- $f(x_i^*) > f(x_i)$: a solução analisada na iteração tem custo superior ao da solução atual. Neste caso, a solução atual pode ser substituída pela solução analisada com probabilidade determinada pela diferença de custos e pela “temperatura” do processo e obtida a partir da distribuição de Boltzman

$$P(\Delta E) = e^{-\frac{\Delta E}{kT}} \quad (4.2)$$

onde $P(\Delta E)$ é a probabilidade do processo de otimização manter a modificação que incorre em um aumento ΔE da função custo. k é o parâmetro do processo (análogo à constante de Stefan–Boltzman) e T é a temperatura instantânea do processo.

A “temperatura” é um parâmetro do recozimento simulado que varia ao longo do processo de acordo com um “programa de resfriamento”. Genericamente, quanto maior for a temperatura, maior a probabilidade do recozimento simulado substituir uma solução de custo inferior por uma solução de custo superior (caminhando assim, na direção do gradiente ascendente de custos). É este o tipo de movimentação que permite ao processo escapar de mínimos locais. À medida que o processo avança, a temperatura é gradativamente reduzida, permitindo assim que este se fixe em uma solução ótima.

A proposta inicial feita por Kirkpatrick et al. (KIRKPATRICK; GELLAT; VECCHI, 1983) era aplicada a problemas combinatórios, como o problema do caixeiro viajante, onde dado um conjunto de cidades que o caixeiro viajante deve percorrer, procura-se pela sequência das cidades que apresenta a menor distância percorrida. Cada sequência representa uma possível solução. Não foi adotada nenhuma métrica para definir a distância entre as duas possíveis soluções. Corana et al. (CORANA et al., 1987) adaptaram o recozimento simulado para que problemas com parâmetros contínuos fossem solucionados. Nesta adaptação foi introduzido o conceito de distância Euclidiana entre possíveis

Figura 49 – Gráfico da temperatura pela energia para o recozimento simulado proposto por Corana et al. (CORANA et al., 1987), onde a próxima solução candidata é selecionada segundo uma distribuição de probabilidades plana e o máximo valor possível diminui com a temperatura.

soluções, uma vez que as soluções estão no espaço real multidimensional. Kirkpatrick et al. (KIRKPATRICK; GELLAT; VECCHI, 1983) já havia observado que o recozimento simulado apresenta principalmente duas fases: fase exploratória e fase de refinamento. Durante a fase exploratória, segundo a proposta feita por Corana et al. (CORANA et al., 1987), o algoritmo permite que a adjacência explorada seja maior; e durante a fase de refinamento, que a adjacência explorada seja menor. Assim, a adjacência varia mantendo a quantidade de soluções aceitas pelo recozimento simulado em um nível razoável, pois soluções candidatas rejeitadas não favorecem para a evolução do algoritmo. Desta maneira a distância Euclidiana explorada diminui com a temperatura (vide Fig. 49).

Ingber (INGBER, 1996) apresentou o ASA (“Adaptive Simulated Annealing”), nesta nova proposta o desvio padrão da densidade de probabilidades das possíveis soluções reduz com a temperatura segundo a distribuição de Cauchy. Assim, o máximo valor possível não diminui com a temperatura, mas a probabilidade de selecionar soluções candidatas distantes da solução atual diminui com a temperatura (vide Fig. 50). Desta forma, quando a temperatura estiver elevada é possível obter praticamente qualquer solução candidata (próxima ou afastada). Mas quando a temperatura cair, a probabilidade de obter uma solução candidata próxima da solução atual aumenta. O desvio padrão da distribuição é determinado pelas derivadas associadas à variável, ou seja, este recozimento simulado requer uma avaliação da derivada.

Membros do Laboratório de Geometria Computacional (MARTINS; TSUZUKI, 2005; MARTINS; TSUZUKI, 2006a; MARTINS, 2007; MARTINS; TSUZUKI, 2007c; MARTINS; TSUZUKI, 2007a; MARTINS; TSUZUKI, 2008a; MARTINS; TSUZUKI, 2008b; MARTINS; TSUZUKI, 2009a; MARTINS; TSUZUKI, 2009b; MARTINS; TSUZUKI, 2010), propuseram um avanço ainda maior que consistia em determinar para cada parâmetro o desvio padrão mais adequado para a sua distribuição de probabilidades na definição

Figura 50 – Gráfico da temperatura pela energia para o recozimento simulado proposto por Ingber (INGBER, 1996), onde a próxima solução candidata é selecionada segundo uma distribuição de probabilidades de Cauchy e o máximo valor possível permanece inalterado com a temperatura.

do próximo candidato. Assim, cada parâmetro pode possuir a sua própria distribuição de probabilidades, independentemente da temperatura (MARTINS; TSUZUKI, 2008b), ou seja, o espaço de soluções aleatórias para a escolha do próximo candidato possui uma vizinhança adaptativa. O principal conceito envolvido é modificar um parâmetro por vez, e identificar se a modificação ao parâmetro específico contribui ou não para que a solução candidata seja aceita. Caso a solução candidata seja aceita, significa que o desvio padrão é relativamente pequeno para este parâmetro e que ele pode ser aumentado. Em caso contrário, onde a solução candidata foi rejeitada, o desvio padrão deste parâmetro específico deve ser reduzido. O algoritmo encontra automaticamente parte da solução que pode ser determinada primeiro, ou que possui prioridade na sua determinação. Particularmente, o problema do empacotamento estudado por membros do Laboratório de Geometria Computacional (MARTINS; TSUZUKI, 2005; MARTINS; TSUZUKI, 2006a; MARTINS, 2007; MARTINS; TSUZUKI, 2007c; MARTINS; TSUZUKI, 2007a; MARTINS; TSUZUKI, 2008a; MARTINS; TSUZUKI, 2008b) pertence a esta classe de problemas, onde as peças são posicionadas sequencialmente, de modo construtivo.

O pseudo-código para o recozimento simulado com vizinhança adaptativa desenvolvido dentro do Laboratório de Geometria Computacional está apresentado no Algoritmo 17. T é a temperatura do sistema e é utilizada como um dos parâmetros de parada global – $T > T_{\text{congelamento}}$, além de definir a probabilidade de aceite de uma solução cujo erro seja maior que o erro da solução correntemente aceita; x é a variável vetor solução; k é índice da temperatura; i é o índice de um coeficiente dentro do vetor solução; c_i é o fator de cristalização de um coeficiente e define a vizinhança adaptativa; v_i é o valor do coeficiente i ; ΔE é a variação na função objetivo causada pela solução candidata; k_b é a constante de Stefan-Boltzman e α é o fator de caimento da temperatura. O critério de parada local está relacionado a uma quantidade de soluções candidatas aceitas e a um total de avaliadas.

Algorithm 17 Recozimento Simulado com Vizinhaça Adaptativa

```

 $\mathbf{x}_0 \leftarrow \text{<Solução inicial aleatória>}$ 
 $k \leftarrow 0$ 
while <Critério de parada global não satisfeito> do
   $k \leftarrow k + 1$ 
  while <Critério de parada local não satisfeito> do
     $i \leftarrow \text{random}(0, 1) * n$ 
     $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + \frac{1}{c_i} \sum \text{random}(-1/2, 1/2) \cdot \mathbf{v}_i$ 
     $\Delta E = F(\mathbf{x}_{k+1}) - F(\mathbf{x}_k)$ 
    if  $\Delta E < 0$  then
       $\mathbf{x}_k \leftarrow \mathbf{x}_{k+1}$ 
       $c_i \leftarrow \text{<realimentação positiva>}$ 
    else
      if  $\text{random}(0, 1) < e^{-\Delta E / k_b T_k}$  then
         $\mathbf{x}_k \leftarrow \mathbf{x}_{k+1}$ 
         $c_i \leftarrow \text{<realimentação positiva>}$ 
      else
         $c_i \leftarrow \text{<realimentação negativa>}$ 
      end if
    end if
  end while
   $T_{k+1} \leftarrow T_k * \alpha$ 
end while

```

4.2 Aplicando o Recozimento Simulado à TIE

O problema inverso da TIE pode ser formulado como um problema de otimização, e para tal, pode ser resolvido com o recozimento simulado. Herrera et al. (HERRERA et al., 2007) minimizaram a função objetivo (4.1) com o recozimento simulado, reconstruindo distribuições de condutividade de um corpo, mas a um custo computacional muito elevado. Isto não é surpreendente, pois cada passo do recozimento simulado envolve a solução de um problema direto da TIE completo, de modo a avaliar a função objetivo.

O problema de otimização utilizado adota a função objetivo apresentada em (4.1), somando-se os erros obtidos através do problema direto para cada padrão de corrente. Além disso, também utiliza a técnica de avaliação parcial da função objetivo, proposta por Martins et al. (MARTINS; TSUZUKI, 2011).

4.2.1 Regularização através da Norma do Gradiente

O problema inverso da TIE é mal posto, e a forma mais comum para contornar isso é através de regularização, utilizando informações *a priori* a respeito do objeto sendo estudado. Na TIE, esta informação pode ser um mapa anatômico obtido, por exemplo, a partir de imagens de RM ou TC, em conjunto a condutâncias conhecidas dos tecidos e também do nível de ruído de sistema de medição utilizado (VAUHKONEN et al., 1998).

Martins et al. (MARTINS; FERNANDES; TSUZUKI, 2014) têm utilizado a regularização através da norma do gradiente (MOJABI; LOVETRI, 2009), obtendo bons resultados. Nesta abordagem, utiliza-se uma função de regularização σ_R^{NG} que penaliza soluções com componentes de alta frequência, favorecendo soluções com uma distribuição de condutância mais suave. A formulação do termo de regularização é

$$\sigma_R^{NG} = \int_{\Omega} \|\nabla \sigma\|^2 dA \quad (4.3)$$

onde Ω é o domínio, ∇ é o operador gradiente e σ é a distribuição de condutividade.

Em um modelo MEF discreto, a distribuição de condutividades $\sigma(x)$ é aproximada pelas funções de interpolação de tal maneira que $\sigma(x) = \sigma^T \cdot \Psi(x)$, onde σ é o vetor solução de condutividades, e $\Psi(x)$ é o vetor de funções base. Assim, o termo de regularização σ_R^{NG} pode ser escrito como

$$\sigma_R^{NG} = \sigma^T \cdot B \cdot \sigma \quad (4.4)$$

sendo que B é formado por elementos $B_{ij} = \int_{\Omega} \nabla \psi_i \psi_j dA$. Se as funções de interpolação utilizadas para $\Psi(x)$ forem as mesma utilizadas para interpolar o potencial elétrico ϕ (vide seção 2.6.2), então a matriz $B_{n \times n}$ é muito semelhante à matriz de rigidez K do modelo MEF da seção 2.5.

Esta função de regularização também é bastante favorável para a implementação paralela, uma vez que o seu cálculo resume-se a uma multiplicação matriz $\times$ vetor, seguida de uma produto interno.

4.3 Heurística de Fora para Dentro

No caso particular da TIE, existem mais informações associadas ao contorno, pois as correntes são aplicadas ao contorno e as tensões são medidas sobre o contorno. Assim, as condutividades elétricas próximas ao contorno devem ser determinadas primeiro. No recozimento simulado utilizado, as condutividades elétricas próximas ao contorno devem ser determinadas na fase de refinamento antes que as condutividades elétricas interiores sejam determinadas. Somersalo et al. (SOMERSALO et al., 1991) propuseram um algoritmo não iterativo que utilizou este conceito. O conceito de que existem mais informações associadas ao contorno é interessante, entretanto ele deve estar associado a uma técnica iterativa, como o recozimento simulado.

A Fig. 51 ilustra esta hierarquia de prioridades, como existem mais informações associadas ao contorno, o recozimento simulado deve processá-las com maior prioridade. O algoritmo deve caminhar do contorno para o interior. Assim, as camadas devem ser priorizadas de acordo com a sua cor: verde, azul, vermelho, amarelo, cor de rosa e laranja. O método proposto por Somersalo et al. (SOMERSALO et al., 1991) não produziu bons resultados pois ele não é iterativo, mesmo a camada verde é influenciada pelo interior

Figura 51 – Exemplo de malha de elementos finitos, representando a hierarquia das informações. No contorno tem-se mais informações e estas condutividades devem ser estimadas primeiro.

sendo necessário retornar e processá-la novamente. Esta é uma atividade que o recozimento simulado realiza de modo efetivo.

4.3.1 Utilizando dados experimentais

Na execução desta etapa, foi utilizado um modelo simples, composto por um recipiente cilíndrico feito de acrílico medindo 300 mm de diâmetro, com 32 eletrodos igualmente espaçados na parede exterior. Os eletrodos são prismáticos, ou seja, suas seções transversais horizontais são invariáveis. O recipiente foi preenchido com água salina até uma altura de 25 mm. A corrente foi aplicada ao modelo com padrões “pula três”, isto é, a corrente foi aplicada entre dois eletrodos que estão separados por quatro posições (isto é, separados por três eletrodos). Esses padrões foram aplicados a cada um dos 32 eletrodos. A corrente aplicada foi de cerca de 10 mA (valores reais variados em cada aplicação) AC em 125 Hz. A fim de produzir fantasmas observáveis, três fatias de pepino, ligeiramente mais espessa do que a altura da solução (mas não o suficiente para produzir efeitos 3D significativos) foram inseridas em um padrão triangular como pode ser visto nas Figs. 52 e 53. Os dados foram coletados tanto para a solução salina vazia como para a solução com fatias de pepino.

Para o modelo de simulação, um modelo de MEF foi criado com 750 elementos triangulares e 450 nós. Observando os dados coletados com o meio vazio, tornou-se óbvio que a impedância de cada eletrodo varia significativamente. Em um primeiro passo, para obter com precisão a impedância de cada eletrodo, o processo de reconstrução foi executado com os dados para o meio vazio, presumido com condutividade uniforme (mas desconhecida)

Figura 52 – Representação esquemática do aparato experimental.

Figura 53 – Foto do aparato experimental.

dentro de um modelo simulado e valores de condutividade independentes para cada eletrodo (que produz um problema de otimização com $32 + 1$ incógnitas).

Para reconstruir os fantasmas de pepino, a distribuição de condutividade no modelo simulado foi parametrizada usando as mesmas funções de primeira ordem de interpolação utilizadas para o potencial do MEF (os parâmetros sendo o valor de condutância em cada nó). A condutância dos elementos do “anel externo” são presumidos uniformes. A malha pode ser vista na Fig. 54.

4.3.2 Forçando a caminhada de fora para dentro

Na implementação feita por Martins (TSUZUKI; MARTINS; TAKASE, 2006; MARTINS; TSUZUKI, 2006b; MARTINS; TSUZUKI, 2007b; MARTINS; TSUZUKI, 2007c; MARTINS, 2007; MARTINS; TSUZUKI, 2008b), a escolha do parâmetro a ser modificado é feita de maneira aleatória, com uma densidade de probabilidade constante para a seleção do elemento. Já foi observado que a distribuição de impedâncias é reconstruída do exterior para o interior, confirmando que existe maior sensibilidade nos elementos mais

Figura 54 – Malha utilizada na reconstrução de imagens a partir dos dados experimentais.

externos. Assim, uma heurística de fora para dentro pode aumentar o número de candidatos aceitos, reduzindo o número de iterações. Tal heurística é possível de ser implementada.

Com o intuito de dar prioridade aos elementos externos em temperaturas altas, e aos elementos internos nas temperaturas baixas, e considerando-se o formato da malha, optou-se pela utilização de camadas circulares, todas com a mesma espessura, como mostrado na Fig. 51. Cada camada tem sua probabilidade de ser sorteada, e dentro de cada camada, todos os nós possuem a mesma probabilidade de sorteio. Supondo uma probabilidade inicial de cada camada igual ao seu número de elementos, tem-se a implementação original, com distribuição de probabilidades plana.

Com o objetivo de forçar que os sorteios dos elementos caminhem de fora para dentro, as camadas mais externas são iniciadas com probabilidades maiores, enquanto as camadas mais internas são iniciadas com probabilidades menores. As probabilidades de cada camada são alteradas dependendo do resultado da iteração do recozimento simulado, aceito ou rejeitado. Caso o candidato tenha sido aceito, deseja-se explorar mais a região em que se encontra, em busca de novos candidatos que possam ser aceitos, então aumenta-se a probabilidade de sua camada. Caso o candidato tenha sido rejeitado, deseja-se buscar um outro candidato mais interno, então reduz-se a probabilidade da sua camada, e aumenta-se a probabilidade da camada imediatamente mais interna. O pseudo-código representando essa heurística está ilustrado no Algoritmo 18.

Na Fig. 55 está ilustrada a variação dos pesos de cada camada conforme a variação da temperatura (observa-se que a temperatura inicial é 0,1, e passa a decrescer). Pode-se observar que os pesos de cada camada elevam-se, até a fase em que passam a ser mais explorados, e depois caem, conforme seus parâmetros já não apresentam bons candidatos.

Algorithm 18 Pseudo-código da heurística de fora para dentro.

```

for cada iteração do
  if candidato aceito then
    aumenta probabilidade de sorteio da camada selecionada
  else
    reduz a probabilidade de sorteio da camada selecionada
    aumenta probabilidade de sorteio da camada imediatamente interior
  end if
end for

```

Figura 55 – Variação dos pesos para sorteio de cada camada, conforme a temperatura.

Na Fig. 56, o mesmo gráfico, normalizado, para fornecer as probabilidades. Na Fig. 57, temos o gráfico do número de sorteio de cada camada, que acompanha o gráfico das probabilidades. O resultado final do processamento está apresentado na Fig. 58.

O impacto da heurística de fora para dentro no desempenho do algoritmo de reconstrução pode ser visto na Fig. 59. Nela estão apresentados os gráficos do número médio de iterações do algoritmo dos Gradientes Conjugados utilizados na solução do sistema linear, para cada temperatura, com e sem a heurística de fora para dentro. Mesmo sem a heurística de fora para dentro, e considerando que o sistema possui mais de 450 nós, é interessante notar que o algoritmo é capaz de obter os resultados utilizando menos de 35 iterações de Gradientes Conjugados. A adoção da heurística de fora para dentro reduz ainda mais esse número de iterações, pela maior parte do processamento.

Como é mostrado na Fig. 60, há também um aumento na taxa de aceite de soluções candidatas. Considerando que o número de candidatos aceitos define o tamanho do laço interno, o aumento na taxa de aceite leva a uma redução no tempo de processamento total. De fato, no computador utilizado, com processador Intel i7 2.8 GHz, temperaturas inicial de 1,0 e final de 0,0001, a imagem na Fig. 58 foi reconstruída em cerca de 205 minutos

Figura 56 – Variação das probabilidades de sorteio de cada camada, conforme a temperatura.

Figura 57 – Variação do sorteio de cada camada conforme a temperatura.

quando utilizada a heurística de fora para dentro aplicada ao algoritmo serial com Eigen. Sem ela, o processo de reconstrução precisou de cerca de 220 minutos, ou seja, 7,3% mais tempo. Como se pode ver, apesar de ser fortemente condicionado pela malha grossa, a distribuição de condutância reconstruída é uma imagem razoável do fantoma físico.

Este estudo indica que a heurística de fora para dentro pode aumentar a velocidade do recozimento simulado. Mais pesquisas podem ser feitas para aumentar o impacto da heurística.

Figura 58 – Imagem reconstruída a partir dos dados experimentais, utilizando a heurística de fora para dentro. Escala apresenta valores de potencial elétrico, em mV.

Figura 59 – Iterações de Gradientes Conjugados por temperatura, para a reconstrução a partir de dados experimentais - com e sem a heurística de fora para dentro.

Figura 60 – Taxa de aceite de candidatos por temperatura - com e sem a heurística de fora para dentro.

5 Paralelização do Problema Inverso da TIE usando GPU

Neste capítulo, será discutida a paralelização do problema inverso da TIE, apresentado no Capítulo 4. Como visto, o problema inverso da TIE pode ser abordado como um problema de otimização, no qual são solucionados diversos problemas diretos da TIE. No Capítulo 3 foi discutida a paralelização do problema direto da TIE, apresentando-se um novo formato para representar matrizes esparsas e estratégias para se paralelizar eficientemente o algoritmo dos Gradientes Conjugados. Todos estes resultados serão utilizados.

5.1 Recozimento Simulado Paralelizado

Como visto na Seção 4.2, a função objetivo para o problema inverso da TIE abordado como um problema de otimização é a soma da norma L2 dos erros entre os potenciais medidos e calculados para cada um dos 32 padrões de corrente. Analisando o Algoritmo 17. Na forma serial, isso é feito resolvendo-se sequencialmente o problema direto e calculando-se o erro para cada um dos padrões de corrente. Isso está ilustrado na Fig. 61.

Uma forma de paralelizar o problema inverso é paralelizar o cálculo do erro para cada um dos 32 padrões de corrente. Até a versão 7 do framework CUDA, a API disponibilizava um método para que cópias de memória pudessem ser realizadas ao mesmo tempo que a execução de kernels através de *streams* distintos. A partir da versão 7, o mecanismo de streams foi expandido para permitir que múltiplos kernels sejam executados concorrentemente.

CUDA streams são usados para gerenciar a concorrência entre unidades de execução em um nível mais alto que a de threads dentro de um kernel, ou seja, com uma maior granularidade. Todas as operações enviadas para serem executadas em um dado stream são executadas sequencialmente, na ordem em que foram enviadas. No caso do problema direto da TIE, isso é exatamente o desejado, uma vez que, idealmente, os kernels são enviados pela CPU para a GPU mais rapidamente do que eles são executados e eles precisam ser executados na exata ordem enviada.

No caso do problema inverso, porém, o problema direto relacionado a cada padrão de corrente é independente dos outros, o que significa que os kernels relacionados aos problemas diretos de cada padrão de corrente podem ser enviados para serem executados sequencialmente em streams diferentes. Isso não significa que todos os kernels de padrões de corrente seriam executados, efetivamente, em paralelo. O nível de execução em paralelo ainda é limitado pelos recursos disponíveis do dispositivo utilizado comparado ao

Figura 61 – Diagrama de sequência para a implementação serial do recozimento interno (ver Algoritmo 17).

paralelismo necessário para cada kernel. Assim, se um dispositivo possui 2.048 núcleos disponíveis, como é o caso da Geforce GTX 980, e os kernels são lançados em blocos de 512 threads, é possível executar até 4 kernels em streams paralelos. Ou, caso um kernel seja lançado usando 4 blocos de 512 threads, o dispositivo já estará “cheio”, não possibilitando paralelização adicional através de streams.

Uma proposta para aumentar o paralelismo e, conseqüentemente, o desempenho do problema inverso é pré-calcular a próxima solução aleatória em CPU enquanto a GPU está encarregada de processar os 32 problemas diretos. É necessário pré-calcular duas soluções aleatórias: uma solução aleatória caso a solução candidata em processamento seja aceita, e outra caso a solução candidata em processamento seja rejeitada. Além disso, é necessário calcular sua decomposição de Cholesky incompleta e realizar a transferência de memória – o que ocorre de forma paralela sem afetar o desempenho do processamento dos kernels. E, como visto na Seção 3.7, o cálculo do pré-condicionador é mais rápido em CPU e é rápido o suficiente para que seu processamento em CPU conclua antes que o problema direto paralelizado em GPU termine.

Assim, a paralelização do recozimento simulado proposta utiliza o laço interno ilustrado na Fig. 62. O laço interno inicia com uma etapa de sincronização para que o cálculo da função objetivo somente seja realizado após o término de todos os problemas diretos. Após a avaliação da função objetivo, as variáveis para a matriz de rigidez e o pré-condicionador a serem utilizados são trocadas na GPU para utilizarem versões pré-calculadas anteriormente, para o caso adequado – solução candidata aceita ou rejeitada. Em seguida, inicia-se o processamento dos 32 problemas diretos em GPU e o pré-cálculo da próxima solução para ambos os casos – aceite ou rejeição, em CPU. Uma proposta adicional é a de utilizar threads separadas para o lançamento de cada problema direto em streams diferentes, com o objetivo de diminuir a sobrecarga da API CUDA, e encher mais rapidamente a fila de execução da GPU. Além disso, um efeito adicional da sobreposição de chamadas da API é a sobreposição das latências de chamadas com chamadas efetivamente, isto é, enquanto a chamada de API em uma thread está no instante de latência, a de outra thread já está em execução. Desta forma, reduz-se as chances de gargalo de execução na CPU.

5.2 Resultados

Para a avaliação da paralelização problema inverso utilizando o formato pJDS colorido, o algoritmo desenvolvido foi testado em um computador equipado com a placa GPU NVIDIA GeForce GTX980, com 2.048 núcleos CUDA, 4GB DDR5 de memória DRAM, utilizando Compute Capability 5.2 e CUDA 7.5. A CPU é um Intel Xeon E5645 2,4GHz com 32GB RAM, rodando Windows 7 Professional. Todos os kernels utilizam

Figura 62 – Ilustração do laço interno proposto para a paralelização do recozimento interno (ver Algoritmo 17).

precisão simples para dados em ponto flutuante, e são lançados em blocos de 512 threads para malhas pequenas – até cerca de 2.500 nós – e 1.024 threads para as demais. O algoritmo apresentado foi comparado a uma implementação serial, Eigen2. Todos os tempos foram medidos considerando cinco diferentes malhas de tamanhos similares, e os tamanhos mostrados representam as médias de tais malhas. O pré-condicionador utilizado foi a decomposição de Cholesky incompleta. O fator de decaimento da temperatura do recozimento simulado utilizado foi 0,9. O critério de parada local utilizado foi de 3.000 soluções aceitas, ou 6.000 soluções totais. A temperatura inicial foi ajustada para que o algoritmo iniciasse com cerca de 50% de soluções aceitas.

Considere um experimento com pepinos como o apresentado na Fig. 63. A reconstrução utilizando o algoritmo serial Eigen com uma malha de 1047 nós está apresentada na Fig. 64. A Fig. 65 apresenta a quantidade de iterações dos Gradientes Conjugados com avaliação incompleta da função objetivo (ver Apêndice B) por temperatura. Pode-se observar que o número de iterações está limitada em aproximadamente 32. Assim, para o problema inverso paralelizado, considerando-se esse experimento, o número de iterações dos Gradientes Conjugados será limitado em 64.

Para o problema inverso massivamente paralelizado proposto, a Fig. 66 apresenta o decaimento da função objetivo ao longo do decaimento da temperatura, enquanto a Fig. 67 apresenta a taxa de aceites de soluções ao longo do decaimento da temperatura.

Figura 63 – Experimento com três pepinos dispostos em padrão triangular.

Figura 64 – Reconstrução de imagem de TIE com três pepinos, utilizando a implementação serial. Escala apresenta valores de potencial elétrico, em mV.

Figura 65 – Iterações dos Gradientes Conjugados com avaliação incompleta da função objetivo por temperatura.

Figura 66 – Decaimento da função objetivo pela temperatura.

Como pode-se observar, a função objetivo possui uma queda muito rápida, tendendo a valores próximos de 24, sem chegar a 0. O resultado do problema inverso massivamente paralelizado está apresentado na Fig. 68. Apesar da quantidade de artefatos, é possível observar a presença dos três pepinos, com a mesma qualidade da imagem reconstruída usando o algoritmo serial.

A Fig. 69 apresenta a linha do tempo das chamadas à API CUDA, com a utilização de uma thread para cada padrão de corrente para a invocação dos kernels do problema direto. O efeito é o enchimento mais rápido da fila de execução. A Fig. 70 apresenta a

Figura 67 – Taxa de aceite de soluções pela temperatura.

Figura 68 – Reconstrução de imagem de TIE com três pepinos, utilizando a implementação massivamente paralelizada proposta. Potencial elétrico em mV.

Figura 69 – Linha do tempo de chamadas da API CUDA, ilustrando a utilização de uma thread para cada padrão de corrente para a invocação de kernels.

Tabela 8 – Comparação do tempo de execução da avaliação de função objetivo, considerando 30 iterações dos Gradientes Conjugados – tempo médio (em ms)

Tamanho	Eigen2	CUDA + CPU
1.046	38,25	30,63
2.590	95,75	35,45
4.450	174,00	42,30
6.900	279,50	55,27
9.500	350,50	73,08
15.100	634,75	119,71
20.700	843,00	162,56

linha do tempo de execução dos 32 streams executando kernels para o problema direto. Os kernels aparecem sobrepostos, mas o agendador de threads do dispositivo é responsável por fazer o chaveamento e balanceamento de threads. Porém, é possível observar um efeito benéfico do uso de múltiplos streams: com a sobreposição de kernels, há redução de momentos de ociosidade.

Com isso, os resultados apresentados na Tabela 8 foram obtidos. Nela, estão apresentados os tempos para uma avaliação da função objetivo para diversos tamanhos de malhas. Como pode-se observar, mesmo para a menor malha a abordagem proposta é vantajosa, considerando o mesmo número de iterações dos Gradientes Conjugados. O ganho em desempenho para diversas malhas está apresentado na Fig. 71. Pode-se observar uma saturação do ganho em torno de 5 vezes.

A Tabela 9 apresenta os tempos de processamento exclusivos da regularização. A regularização é feita em GPU para aproveitar as vantagens da paralelização da multiplicação matriz esparsa $\times$ vetor e produto interno, operações utilizadas para o cálculo da

Figura 70 – Linha do tempo de execução dos kernels do problema direto, ilustrando a utilização de um stream para cada padrão de corrente.

Figura 71 – Ganho de desempenho no problema inverso utilizando a abordagem proposta, em relação ao número de nós.

Tabela 9 – Comparação do tempo de processamento para a regularização – tempo médio (em μs) efetivo para a CPU

Tamanho	Eigen2	CUDA
1.046	21,62	23,58
2.590	59,77	26,63
4.450	104,62	33,81
6.900	157,92	38,89
9.500	208,77	44,30
15.100	376,85	64,59
20.700	486,07	78,71

Tabela 10 – Comparação do tempo de processamento e transferência de memória do pré-cálculo da próxima solução – tempo médio (em ms)

Tamanho	Pré-cálculo
1.046	0,26
2.590	0,50
4.450	0,76
6.900	1,15
9.500	1,75
15.100	3,19
20.700	4,51

regularização através da norma do gradiente, como explicado na Seção 4.2.1. A regularização depende apenas da solução candidata, e pode ser executada em paralelo aos problemas diretos relacionados aos 32 padrões de corrente, em um stream próprio. Como pode-se observar, os tempos de processamento da regularização são muito menores do que o tempo de processamento do problema direto. Também nota-se que a sobrecarga de chamada de kernel impede que a regularização em CUDA seja mais rápida que a serial para malhas pequenas.

A Tabela 10 apresenta os tempos de processamento do pré-cálculo para cada uma das próximas soluções, para os casos de aceite e de rejeição da solução sendo avaliada. Ambas as novas soluções podem ser pré-calculadas em paralelo, na CPU, sendo que apenas a transferência para a memória da GPU envolve o dispositivo gráfico. Como pode-se observar, os tempos são muito inferiores do que o tempo de processamento para os problemas diretos para os 32 padrões de corrente. Assim, em conjunto com o tempo para a regularização, pode-se considerar que cada laço interno do recozimento simulado leva o tempo apresentado na Tabela 8.

5.3 Conclusão

Neste capítulo, foi estudada a paralelização do recozimento simulado para a resolução do problema inverso da TIE, e foi proposta uma abordagem híbrida de paralelização, com partes do algoritmo em CPU e partes em GPU. Esta abordagem mostrou um ganho efetivo de desempenho, maior do que apenas o ganho observado no problema direto. Isso foi obtido utilizando-se múltiplas threads para enviar kernels para serem executadas em streams distintos, reduzindo a ociosidade da GPU entre kernels e maximizando sua utilização.

A opção por realizar operações cuja paralelização foi pouco efetiva em CPU mostrou-se acertada. Desta forma, a GPU pode ficar dedicada ao cálculo da função objetivo, de forma massivamente paralelizada, enquanto a CPU se encarrega do controle do recozimento simulado e de operações de preparação para a próxima iteração, como o pré-cálculo das novas soluções para ambas as possibilidades – aceite ou rejeição.

Verificou-se que apesar dos esforços para diminuir as sobrecargas e as latências apresentados no Capítulo 3, a utilização de streams fornece um ganho adicional de desempenho.

Assim, conclui-se que a abordagem híbrida massivamente paralelizada proposta fornece ganhos reais em termos de desempenho para o problema inverso da TIE.

6 Conclusões e Trabalhos Futuros

O principal objetivo desta tese é reduzir o tempo computacional da reconstrução de imagens da TIE utilizando o recozimento simulado. Para isso, foram propostas (i) uma nova heurística para seleção de candidatos, que potencializa a convergência “de fora para dentro” já observada, (ii) uma nova representação de matrizes esparsas que permite a paralelização massiva de algoritmos utilizados no problema da TIE e (iii) uma abordagem híbrida para o problema inverso, utilizando paralelamente a CPU e a GPU para funções distintas.

A heurística de fora para dentro apresentou efeitos de redução na quantidade de iterações necessárias dos Gradientes Conjugados para a avaliação parcial da função objetivo. Além disso, também apresentou um aumento na taxa de aceite de soluções candidatas, com o resultado final sendo uma redução no tempo computacional para o problema inverso da TIE. No entanto, essa redução mostrou-se muito pequena para ser considerada um fator importante. Acredita-se, porém, que novos estudos podem aumentar o ganho causado por esta heurística, com o aumento na qualidade de seleção de soluções candidatas.

A nova representação de malha proposta apresentou vantagens sobre os formatos mais comumente utilizados. Sendo um formato especializado para as características apresentadas pelo framework CUDA, foca-se em evitar acessos desalinhados à memória ao agrupar todos os endereços nos vetores de dados, preencher todos os kernels de forma igual para que não haja divergência de threads, representar por coluna para se obter acessos contíguos à memória global dentro de warps, reposicionar os valores da diagonal principal – muito utilizados nos solvers triangulares e no cálculo do pré-condicionador, e, principalmente, em utilizar cores para representar e agrupar as dependências intrínsecas ao modelo MEF, que impedem a paralelização direta de algoritmos como o solver triangular. Todas essas características resultaram em uma sobrecarga de acesso aos dados que a torna custosa para malhas muito pequenas, mas que rapidamente apresentam vantagens conforme o crescimento das matrizes.

Esta nova representação de matriz esparsa, porém, tem como principal vantagem possibilitar a paralelização completa do algoritmo dos Gradientes Conjugados em GPU. Os resultados apresentados demonstram que todos os kernels estudados apresentam vantagens em termos de desempenho sobre seus correspondentes seriais, a partir de um determinado tamanho da malha. Infelizmente, a sobrecarga e a latência de chamadas da API CUDA demonstraram-se altas demais para que o algoritmo fosse viável para as menores malhas em sua forma mais modularizada. Com a fusão de todas as operações, respeitando-se requisitos de sincronização para que as operações ocorressem todas na ordem correta,

observou-se que mesmo para a menor das malhas estudadas o método de paralelização proposto apresenta ganhos efetivos em desempenho, com uma saturação próxima a 4 vezes mais rápido a partir de, aproximadamente, 12.500 nós. Ressalta-se, ainda, que os resultados apresentados não utilizaram o melhor algoritmo para coloração de grafos – algo que não fazia parte do escopo deste trabalho, indicando que os ganhos podem ser ainda maiores do que os observados.

Uma abordagem híbrida foi proposta para a resolução do problema inverso da TIE, com partes do algoritmo de recozimento simulado sendo realizado em GPU – o cálculo dos 32 problemas diretos ligados aos padrões de corrente e o cálculo da regularização – e partes em CPU – o pré-cálculo das próximas soluções, além do próprio controle do recozimento simulado. Além disso, foi proposta a utilização de múltiplas threads para a realização dos acionamentos independentes relacionados aos padrões de corrente e à regularização, com o lançamento de kernels em streams de processamento distintos. Isso permite que o ganho de desempenho para o problema inverso seja superior ao observado no problema direto – saturação em cerca de 5 vezes mais rápido a partir de aproximadamente 7.000 nós, devido à redução de períodos de ociosidade entre kernels ou quando a GPU estava aguardando instruções da CPU. O pré-cálculo das duas possibilidades de próxima solução também efetivamente diminui o tempo de ociosidade da GPU, uma vez que o processamento da próxima solução pode iniciar tão logo for concluída a avaliação da solução candidata corrente, evitando-se o tempo de montagem de matriz, cálculo do pré-condicionador e transferência de memória entre CPU e GPU.

Observa-se que estes resultados para o problema inverso massivamente paralelizado não utilizam as duas técnicas de aceleração apresentadas para o problema inverso serial: a avaliação incompleta da função objetivo e a heurística de fora para dentro. Parte do resultado da avaliação completa da função objetiva foi utilizado para a determinação do número de iterações dos Gradientes Conjugados necessário para a convergência, mas reconhece-se que isso acaba por reduzir o ganho efetivo em desempenho, já que é realizado, em média, um número maior de iterações dos Gradientes Conjugados para a avaliação da função objetivo. Um trabalho futuro importante é inclusão da avaliação parcial da função objetivo. Acredita-se que, como para realizar a avaliação incompleta da função torna-se necessário manter ambas as soluções – a candidata e corrente – em memória e em processamento, talvez a utilização de duas placas de vídeo seja uma solução interessante. O uso de duas placas de vídeo também beneficiaria grandemente o cálculo atual, já que os 32 padrões de corrente são problemas independentes entre si e poderiam ser divididos entre dois dispositivos.

Ressalta-se que o método de paralelização massiva proposto nesta tese não restringe em qualquer forma o formato da malha. Assim, este mesmo método pode ser aplicado a modelos complexos – como os que consideram efeitos capacitivos – ou tridimensionais. No

caso tridimensional, em particular, considerando-se que o número de cores será provavelmente muito maior do que os vistos neste trabalho, recomenda-se criar uma malha que otimize a paralelização, com a redução do número de cores, mas levando-se em conta o erro de modelagem. Neste trabalho, foi realizado o estudo do erro de modelagem para o caso bidimensional. Um estudo semelhante deve ser feito para o caso tridimensional.

Por fim, é interessante comentar a utilização de um framework que tente generalizar a programação em GPU como o CUDA. As abstrações realizadas de forma a tornar a programação mais próxima do que a realizada em CPU apresentou custos elevados em termos de desempenho, com latências comparáveis aos tempos de execução dos próprios kernels. O ganho em facilidade acaba por reduzir o ganho em desempenho, e o movimento observado atualmente é na redução da abstração nos principais frameworks para computação gráfica usando GPU, como OpenGL e DirectX 12: a redução de chamadas de alto nível, para que, através da utilização de instruções em mais baixo nível, a sobrecarga de chamadas seja reduzido e o framework apresente menores latências (DOBERSBERGER, 2015). Isso torna-se cada vez mais importante com a adoção de resoluções cada vez maiores.

Referências

- ADLER, A.; GUARDO, R. Electrical impedance tomography: regularized imaging and contrast detection. *IEEE Transactions on Medical Imaging*, v. 15, p. 170–179, 1996. Citado na página 22.
- AMATO, M. B. P. et al. Effects of a protective ventilation strategy on mortality in the acute respiratory distress syndrome. *The New England Journal of Medicine*, v. 338, p. 347–354, 1998. Citado 2 vezes nas páginas 18 e 26.
- AVIS, N. J.; BARBER, R. H. D. C. Image reconstruction using non-adjacent drive configuration. *Physiological Measurements*, v. 16, p. 153–160, 1994. Citado na página 23.
- BARBER, D. C. A review of image reconstruction techniques for electrical impedance tomography. *Medical Physics*, v. 16, p. 162–169, 1989. Citado 2 vezes nas páginas 23 e 28.
- BARBER, D. C.; BROWN, B. H. Applied potencial tomography. *Journal of Physics E: Scientific Instruments*, v. 17, p. 723–733, 1984. Citado 4 vezes nas páginas 17, 21, 23 e 25.
- BATES, R.; GARDENM, K. L.; PETERS, R. M. Overview of computerized tomography with emphasis on future developments. *Proceedings of the IEEE*, v. 71, p. 356–372, 1983. Citado na página 21.
- BATHE, K.-J. *Finite Element Procedures (Part 1-2)*. [S.l.]: Prentice Hall, 1995. 1037 p. ISBN 0133014584. Citado 2 vezes nas páginas 28 e 128.
- BERGER, P.; BROUAYE, P.; SYRE, J. C. A mesh coloring method for efficient mimd processing in finite element problems. In: *Proc. of the International Conference on Parallel Processing*. [S.l.]: IEEE Computer Society, 1982. p. 41–46. Citado na página 59.
- BORCEA, L. Electrical impedance tomography. *Inverse Problems*, v. 18, p. 99–136, 2002. Citado 2 vezes nas páginas 22 e 23.
- BROWN, B. H. Imaging with non-ionising radiation. In: *Tissue impedance methods*. [S.l.]: Survery Univ. Press, 1983. p. 85–110. Citado na página 23.
- BROWN, B. H. Medical impedance tomography and process impedance tomography: a brief review. *Measurements Science and Technology*, v. 12, p. 991–996, 2001. Citado na página 22.
- BROWN, B. H.; SEAGAR, A. The Sheffield data collection system. *Clinical Physics and Physiological Measurements*, v. 8, p. A91–A97, 1987. Citado na página 23.
- CASTRO SOBRINHO, A. d. S. *Introdução ao Método dos Elementos Finitos*. Rio de Janeiro: Ltda., Editora Ciência Moderna, 2006. ISBN 85-7393-512-X. Citado 3 vezes nas páginas 28, 128 e 131.
- CENSOR, Y. Finite series expansion reconstruction methods. *Proceedings of the IEEE*, v. 71, p. 409–419, 1983. Citado 2 vezes nas páginas 22 e 27.

- CHENEY, M.; ISAACSON, D. Distinguish ability in impedance imaging. *IEEE Transactions on Biomedical Engineering*, v. 39, p. 852–860, 1991. Citado na página 23.
- CHENEY, M.; ISAACSON, D. An overview of inversion algorithms for impedance imaging. In: SATTINGER, O. H.; TRACY, C. A.; VENAKIDES, S. (Ed.). *Inverse scattering and applications*. [S.l.]: AMS, 1991. p. 85–110. Citado na página 25.
- CHENEY, M.; ISAACSON, D.; NEWELL, J. C. Electrical impedance tomography. *SIAM Review*, v. 41, p. 85–101, 1999. Citado 2 vezes nas páginas 25 e 26.
- CHENEY, M. et al. Noser: an algorithm for solving the inverse conductivity problem. *International Journal of Imaging Systems and Technology*, v. 2, p. 66–75, 1990. Citado 3 vezes nas páginas 22, 26 e 27.
- CLAY, M. T.; FRREE, T. C. Weighted regularization in electrical impedance tomography with applications to accute cerebral stroke. *IEEE Transactions on Medical Imaging*, v. 21, p. 629–637, 2002. Citado na página 18.
- COOK, R. D. et al. Act3: high speed, high precision electrical impedance tomography. *IEEE Transactions on Biomedical Imaging*, v. 41, p. 713–722, 1994. Citado na página 25.
- CORANA, A. et al. Minimizing multimodal functions of continuous variables with the simulated annealing algorithm. *ACM Transactions on Mathematical Software*, v. 13, p. 262–280, 1987. Citado 3 vezes nas páginas 9, 92 e 93.
- DOBERSBERGER, S. *Reducing Driver Overhead in OpenGL, Direct3D and Mantle*. Tese (Doutorado) — University of Applied Sciences Technikum Wien, 2015. Citado na página 117.
- EVANS, L. C. *Partial Differential Equations: Second Edition (Graduate Studies in Mathematics)*. 2nd. ed. Province, Rhode Island: American Mathematical Society, 2010. v. 19. 749 p. (Graduate Series in Mathematics, v. 19). ISBN 0821849743. Citado na página 128.
- EYÜBOGLU, B. M.; BROWN, B. H.; BARBER, D. C. In vivo imaging of cardiac related impedance changes. *IEEE Engineering Med. Biol. Mag.*, v. 8, p. 39–45, 1989. Citado na página 18.
- FUCKS, L. P. Detection and imaging of electrical conductivity and permittivity at low frequency. *IEEE Transactions on Biomedical Engineering*, v. 38, p. 1106–1110, 1991. Citado na página 23.
- GISSER, D. G.; ISAACSON, D.; NEWELL, J. C. Theory and performance of an adaptive tomography system. *Clinical Physics and Physiological Measurements*, v. 9, p. A35–A41, 1988. Citado na página 25.
- GISSER, D. G.; ISAACSON, D.; NEWELL, J. C. Electric current tomography and eigenvalues. *Journal of Applied Mathematics*, v. 50, p. 1623–1634, 1990. Citado na página 22.
- HENDERSON, R. P.; WEBSTER, J. G. An impedance camera for spatially specific measurements of the thorax. *IEEE Transactions on Biomedical Engineering*, v. 25, p. 250–254, 1978. Citado na página 21.

- HERMAN, G. T. *Image reconstruction from projections - the fundamentals of computerized tomography*. San Francisco: Academic Press, 1980. Citado na página 21.
- HERRERA, C. N. L. *Algoritmo de tomografia por impedância elétrica baseado em simulated annealing*. Dissertação (Mestrado) — Escola Politécnica da USP, 2007. Citado 2 vezes nas páginas 26 e 28.
- HERRERA, C. N. L. et al. Electrical impedance tomography algorithm using simulated annealing search method. In: *Proceedings of International Congress Of Mechanical Engineering*. Brasília: ABCM, 2007. Citado na página 95.
- HINSHAW, W.; LENT, A. Nuclear magnetic resonant. *Proceedings of the IEEE*, v. 71, p. 338–350, 1983. Citado na página 21.
- HOLDER, D. *Clinical and Physiological Applications of Electrical Impedance Tomography*. New York: ULC Press, 1993. Citado na página 25.
- HUA, P.; WEBSTER, J. G.; TOMPKINS, W. J. A regularised electrical impedance tomography reconstruction algorithm. *Clinical Physics and Physiological Measurements*, v. 9, p. 137–141, 1988. Citado 2 vezes nas páginas 23 e 127.
- HUA, P. et al. Finite element modeling of electrode-skin contact impedance in electrical impedance tomography. *IEEE Transactions on Biomedical Engineering*, v. 40, p. 335–343, 1993. Citado na página 18.
- HYARIC, A. L. A one step image reconstruction algorithm for electrical impedance tomography in three dimensions. *Physiological Measurement*, v. 21, p. 95–98, 2000. Citado na página 26.
- INGBER, L. Adaptive simulated annealing (asa): lessons learned. *Control and Cybernetics*, v. 25, p. 33–54, 1996. Citado 3 vezes nas páginas 10, 93 e 94.
- ISAACSON, D.; CHENEY, M.; NEWELL, J. C. Comments on reconstruction algorithms. *Clinical Physics and Physiological Measurements*, v. 13, p. 83–89, 1992. Citado na página 23.
- JAUNCEY, G. E. M. The birth and early infancy of the x-rays. *American Journal of Physics*, v. 13, p. 362–379, 1945. Citado na página 21.
- KAPIO, J. P.; SOMERSALO, E. *Statistical and Computational Problems*. 1. ed. [S.l.]: Springer, 2004. Citado na página 37.
- KAO, T. J. et al. A 3D reconstruction algorithm for EIT using a handheld probe for breast cancer detection. *Physiological Measurements*, v. 27, p. S1–S11, 2006. Citado na página 18.
- KERSHAW, S. The incomplete Cholesky-Conjugate Solution of Systems. *Journal of Computational Physics*, v. 65, p. 43–65, 1978. Citado 2 vezes nas páginas 44 e 135.
- KIRKPATRICK, S.; GELLAT, C. D.; VECCHI, M. P. Optimization by simulated annealing. *Science*, v. 220, p. 671–680, 1983. Citado 3 vezes nas páginas 91, 92 e 93.
- KREUTZER, M. et al. Sparse matrix-vector multiplication on GPGPU clusters: A new storage format and a scalable implementation. In: *IPDPS Workshops*. [S.l.: s.n.], 2012. p. 1696–1702. ISBN 978-1-4673-0974-5. Citado na página 55.

- KUMAR, P.; KUMAR, M.; BASU, A. Parallel algorithms for sparse triangular system solution. *Parallel Computing*, v. 19, n. 2, p. 187 – 196, 1993. ISSN 0167-8191. Citado na página 56.
- LAKSHMANAN, H. D. S. Simultaneous parameter estimation and segmentation of gibbs random fields using simulated annealing. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, v. 11, p. 799–813, 1989. Citado na página 133.
- LAWRENCE, C. W. *Electronics for real-time and three dimensional electrical impedance tomography*. Tese (Doutorado) — University of Denver, 1996. Citado na página 23.
- LEWITT, R. M. Reconstruction algorithms: transform methods. *Proceedings of the IEEE*, v. 71, p. 390–408, 1983. Citado 2 vezes nas páginas 22 e 27.
- LIMA, C. *Estudo da obtenção de imagens de tomografia de impedância elétrica do pulmão pelo método de otimização topológica*. Tese (Doutorado) — Escola Politécnica da USP, 2006. Citado 2 vezes nas páginas 26 e 27.
- LIMA, C.; SILVA, E. C. N.; GONZALES-LIMA, R. Topology optimization method applied to obtain images from electrical impedance tomography technique. In: *Proceedings of the 17th International Conference of Mechanical Engineering*. [S.l.: s.n.], 2003. p. 535–538. Citado na página 26.
- LIN, C.-Y.; CHUNG, Y.-C.; LIU, J.-S. Efficient data compression methods for multidimensional sparse array operations based on the EKMR scheme. *IEEE Transactions on Computers*, v. 52, n. 12, p. 1640 – 1646, dec. 2003. ISSN 0018-9340. Citado na página 53.
- MARTINS, T. et al. Electrical impedance tomography reconstruction through simulated annealing with incomplete evaluation of the objective function. In: *Proc 33th IEEE EMBS*. Boston, USA: [s.n.], 2011. p. 7033–7036. Citado na página 44.
- MARTINS, T. C. *Estudo do recozimento simulado e do polígono de obstrução aplicados ao problema de empacotamento rotacional de polígonos irregulares não-convexos em recipientes fechados*. Tese (Doutorado) — Escola Politécnica da USP, 2007. Citado 3 vezes nas páginas 93, 94 e 98.
- MARTINS, T. C.; TSUZUKI, M. S. G. Applying the simulated annealing to the problem of positioning rotational non convex polygons. *XVIII International Congress of Mechanical Engineering*, 2005. Citado 2 vezes nas páginas 93 e 94.
- MARTINS, T. C.; TSUZUKI, M. S. G. Simulated annealing applied to the rotational polygon packing. In: *Proceedings of the IFAC Symposium Information Control Problems in Manufacturing*. Saint Ettienne, França: IFAC, 2006. p. 475–480. Citado 2 vezes nas páginas 93 e 94.
- MARTINS, T. C.; TSUZUKI, M. S. G. Simulated annealing applied to the rotational polygon packing. In: *Proceedings of the IFAC Symposium Information Control Problems in Manufacturing*. Saint Etienne: [s.n.], 2006. p. 475–480. Citado na página 98.
- MARTINS, T. C.; TSUZUKI, M. S. G. Irregular rotational placement of shapes over non-convex containers with fixed dimensions. In: *Proceedings of the IFAC Workshop on Intelligent Manufacturing Systems*. Alicante: [s.n.], 2007. Citado 2 vezes nas páginas 93 e 94.

- MARTINS, T. C.; TSUZUKI, M. S. G. Irregular rotational placement of shapes over non-convex containers with fixed dimensions. In: *Proceedings of the IFAC Workshop on Intelligent Manufacturing Systems*. Alicante, Espanha: IFAC, 2007. Citado na página 98.
- MARTINS, T. C.; TSUZUKI, M. S. G. Solving irregular rotational knapsack problems. In: *Proceedings of the Seventh International Conference on Intelligent Systems Design and Applications*. Rio de Janeiro, Brasil: IEEE, 2007. Citado 3 vezes nas páginas 93, 94 e 98.
- MARTINS, T. C.; TSUZUKI, M. S. G. Rotational placement of irregular polygons over containers with fixed dimensions using simulated annealing and no-fit polygons. *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, v. 30, p. 196–203, 2008. Citado 2 vezes nas páginas 93 e 94.
- MARTINS, T. C.; TSUZUKI, M. S. G. Simulated annealing with adaptive neighborhood applied to the placement over containers with fixed dimensions. In: *Proceedings of the IFAC Workshop on Intelligent Manufacturing Systems*. Szczecin, Polônia: IFAC, 2008. Citado 5 vezes nas páginas 93, 94, 98, 135 e 137.
- MARTINS, T. C.; TSUZUKI, M. S. G. Comparison of deterministic heuristics and simulated annealing for the rotational placement problem over containers with fixed dimensions. In: *Proceedings of the 13th IFAC Symposium Information Control Problems in Manufacturing*. Moscow: [s.n.], 2009. Citado na página 93.
- MARTINS, T. C.; TSUZUKI, M. S. G. Placement over containers with fixed dimensions solved with adaptive neighborhood simulated annealing. *Bulletin of the Polish Academy of Sciences Technical Sciences*, v. 57, p. 273–280, 2009. Citado na página 93.
- MARTINS, T. C.; TSUZUKI, M. S. G. Simulated annealing applied to the irregular rotational placement of shapes over containers with fixed dimensions. *Expert Systems with Applications*, v. 37, p. 1955–1972, 2010. Citado na página 93.
- MARTINS, T. C.; TSUZUKI, M. S. G. Simulated annealing with partial evaluation of objective function applied to electrical impedance tomography. In: *Proceedings of the 18th IFAC World Congress*. Milano, Italy: [s.n.], 2011. p. 4989–4994. Citado 3 vezes nas páginas 95, 133 e 137.
- MARTINS, T. de C.; FERNANDES, A. V.; TSUZUKI, M. de S. G. Image reconstruction by electrical impedance tomography using multi-objective simulated annealing. In: *Biomedical Imaging (ISBI), 2014 IEEE 11th International Symposium on*. [S.l.: s.n.], 2014. p. 185–188. Citado na página 96.
- MELLO, L. A. M. et al. Three-dimensional electrical impedance tomography: a topology optimization approach. *IEEE transactions on bio-medical engineering*, v. 55, n. 2 Pt 1, p. 531–40, 2008. Citado 2 vezes nas páginas 44 e 91.
- MEURANT, G. Estimates of the l2 norm of the error in the conjugate gradient algorithm. *Numerical Algorithms*, v. 40, n. 2, p. 157–169, out. 2005. ISSN 1017-1398. Citado 2 vezes nas páginas 133 e 135.
- MEURANT, G. *The Lanczos and Conjugate Gradient Algorithms: From Theory to Finite Precision Computations*. [S.l.]: Society for Industrial and Applied Mathematics, 2006. Citado 2 vezes nas páginas 42 e 43.

- MIRANDA, D. *Tomografia por impedância elétrica utilizando o método de Newton-Raphson*. Dissertação (Mestrado) — Escola Politécnica da USP, 2003. Citado na página 26.
- MOJABI, P.; LOVETRI, J. Overview and classification of some regularization techniques for the gauss-newton inversion method applied to inverse scattering problems. *Antennas and Propagation, IEEE Transactions on*, v. 57, n. 9, p. 2658–2665, Sept 2009. ISSN 0018-926X. Citado na página 96.
- MOLINA, N. A. V. *Redução de erro numérico no filtro estendido de kalman aplicado à tomografia por impedância elétrica*. Dissertação (Mestrado) — Escola Politécnica da USP, 2002. Citado na página 26.
- MOLINA, N. A. V. et al. Redução de erro numérico de kalman estendido aplicado à tomografia por impedância elétrica. In: *Proceedings of the 17th International Conference of Mechanical Engineering*. [S.l.: s.n.], 2003. p. 571–574. Citado na página 26.
- MOTTA, L. *Estudo da obtenção de imagens tridimensionais de tomografia por impedância elétrica pelo método de otimização topológica*. Dissertação (Mestrado) — Escola Politécnica da USP, 2005. Citado na página 26.
- NATH, R. et al. Optimizing symmetric dense matrix-vector multiplication on GPUs. In: *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. New York, NY, USA: ACM, 2011. (SC '11), p. 6:1–6:10. ISBN 978-1-4503-0771-0. Citado na página 48.
- NEWELL, J. C. et al. Assessment of acute pulmonary edema in dogs by electrical impedance imaging. *IEEE Transactions on Biomedical Engineering*, v. 43, p. 133–138, 1996. Citado na página 23.
- NVIDIA. *CUDA C Programming Guide*. 2015. <http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf>. Citado na página 51.
- OLMI, R.; BINI, M.; PRIORI, S. A genetic algorithm approach to image reconstruction in electrical impedance tomography. *IEEE Transactions on Evolutionary Computation*, v. 4, p. 83–88, 2000. Citado na página 27.
- PAULSON, K.; BRECKON, W.; PIDCOCK, M. A hybrid phantom for electrical impedance tomography. *Clinical Physics and Physiological Measurement*, v. 13, p. 155–159, 1992. Citado na página 26.
- PRICE, L. R. Electrical impedance computed tomography: a new ct imaging technique. *IEEE Transactions Nuclear Science*, NS-26, p. 2736–2739, 1979. Citado na página 21.
- RIGAUD, B. et al. Experimental acquisition system for impedance tomography with active electrodes approach. *Medical & Biological Engineering & Computing*, v. 23, p. 593–599, 1993. Citado na página 25.
- SANTOSA, F.; VOGELIUS, M. A backprojection algorithm for electrical impedance imaging. *SIAM J. Appl. Math.*, v. 50, p. 216–243, 1990. Citado na página 26.
- SCHMITT, U. et al. Efficient algorithms for the regularization of dynamic inverse problems: li applications. *Inverse Problems*, v. 18, p. 659–676, 2002. Citado na página 22.

- SEAGAR, A.; BARBER, D.; BROWN, B. Theoretical limits to sensitivity and resolution in impedance imaging. *Clinical Physics and Physiological Measurements*, v. 8, p. 13–31, 1987. Citado na página 23.
- SHEWCHUK, J. R. *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain*. Pittsburgh, PA, USA, 1994. Citado 3 vezes nas páginas 42, 43 e 44.
- SILVA, O. L. *Muscle Contraction Detection Using Electrical Impedance Tomography*. Tese (Doutorado) — Escola Politécnica da USP, 2012. Citado na página 37.
- SOMERSALO, E. et al. Layer stripping: a direct numerical method for impedance imaging. *Inverse Problems*, v. 7, p. 899–926, 1991. Citado na página 96.
- SYLVESTER, J.; UHLMANN, G. A. Global uniqueness theorem for an inverse boundary value problem. *Annals of Mathematics*, v. 125, p. 153–169, 1987. Citado 2 vezes nas páginas 22 e 25.
- TANG, M. X. et al. The number of electrodes and basis function in eit image reconstruction. *Physiological Measurement*, v. 23, p. 129–140, 2002. Citado 2 vezes nas páginas 23 e 25.
- TIKHONOV, A. N. *Ill-posed Problems in Natural Sciences: Proceedings of the International Conference*. Moscow: [s.n.], 1991. Citado na página 22.
- TRIGO, F. C. *Estimação não linear de parâmetros através dos filtros de Kalman na tomografia por impedância elétrica*. Tese (Doutorado) — Escola Politécnica da USP, 2005. Citado na página 26.
- TRIGO, F. C.; LIMA, R. G.; AMATO, M. B. P. A two phases kalman filter algorithm for electrical impedance tomography. In: *Proceedings of the 17th International Conference on Biomedical Applications of Electrical Impedance Tomography*. [S.l.: s.n.], 2003. Citado na página 26.
- TRIGO, F. C.; LIMA, R. G.; AMATO, M. B. P. Electrical impedance tomography using the extended Kalman filter. *IEEE T Biomed Eng*, v. 51, p. 72–81, 2004. Citado 5 vezes nas páginas 17, 18, 26, 27 e 29.
- TSUZUKI, M. S. G.; MARTINS, T. C.; TAKASE, F. K. Robot path planning using simulated annealing. In: *Proceedings of the IFAC Symposium Information Control Problems in Manufacturing*. Saint Etienne, França: IFAC, 2006. Citado na página 98.
- VANEGAS, N. *Propagação de erro numérico no filtro estendido de kalman aplicado à tomografia por impedância elétrica*. Dissertação (Mestrado) — Escola Politécnica da USP, 2002. Citado na página 26.
- VAUHKONEN, M.; KARJAKAINEN, P. A.; KAIPIO, J. P. A kalman filter approach to track fast impedance changes in electrical impedance tomography. *IEEE Transactions on Biomedical Engineering*, v. 45, p. 486–493, 1998. Citado na página 27.
- VAUHKONEN, M. et al. Tikhonov regularization and prior information in electrical impedance tomography. *Medical Imaging, IEEE Transactions on*, v. 17, n. 2, p. 285–293, April 1998. ISSN 0278-0062. Citado na página 95.

VÁZQUEZ, F.; FERNÁNDEZ, J.; GARZÓN, E. M. A new approach for sparse matrix vector product on NVIDIA GPUs. *Concurrency and Computation: Practice and Experience*, v. 23, n. 8, p. 815–826, 2011. Citado 2 vezes nas páginas 53 e 54.

WEST, R. M. et al. Application specific optimization of regularization for electrical impedance tomography. *Measurement Science & Technology*, v. 12, p. 1050–1054, 2001. Citado na página 22.

WILF, H. S. *Mathematics for the Physical Sciences*. New York: Dover Publications, INC., 1962. Disponível em: <http://www.math.upenn.edu/~wilf/website/Mathematics_for_the_Physical_Sciences.html>. Citado na página 43.

YORKEY, T. J.; WEBSTER, J. G.; TOMPKINS, W. J. Comparing reconstruction algorithms for electrical impedance tomography. *IEEE Transactions on Biomedical Engineering*, v. 34, p. 843–852, 1987. Citado 3 vezes nas páginas 22, 25 e 27.

Apêndices

APÊNDICE A – Formulação Matemática da TIE

O problema direto típico da TIE é definido como, dado um domínio Ω , a distribuição de condutividade σ e a corrente $\mathbf{J}$ injetada pelos eletrodos na fronteira do domínio, encontrar a distribuição de tensões ϕ e em particular, os potenciais nos eletrodos ϕ_m (uma formulação alternativa tem os potenciais nos eletrodos ϕ_m determinados e as correntes nos eletrodos como incógnitas). Neste trabalho, possíveis efeitos capacitivos serão ignorados; sabe-se que eles são muito importantes, mas sua modelagem envolveria a especificação do hardware utilizado, o que não faz parte do escopo deste trabalho.

Assim, as equações de Maxwell que governam as propriedades elétricas do meio condutivo podem ser escritas como

$$\nabla \cdot \mathbf{E} = \frac{\hat{\rho}}{\varepsilon_r \varepsilon_0} \quad (\text{A.1})$$

onde ∇ é o operador gradiente, $\mathbf{E}$ é o campo elétrico, $\hat{\rho}$ é a densidade de carga, ε_r e ε_0 são as permissividades relativa e específica, respectivamente. Tem-se também que

$$\nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t} \implies \oint_c \mathbf{E} \cdot d\mathbf{l} = -\frac{d}{dt} \int_s \mathbf{B} \cdot d\mathbf{A} \quad (\text{A.2})$$

onde $\mathbf{B}$ é a densidade de fluxo magnético. Hua et al. (HUA; WEBSTER; TOMPKINS, 1988) mostram que quando a frequência da corrente que passa pelo meio é inferior a 30MHz o termo contendo derivada de densidade de fluxo magnético é nulo. Assim

$$\nabla \times \mathbf{E} = 0 \implies \oint_c \mathbf{E} \cdot d\mathbf{l} = 0. \quad (\text{A.3})$$

Em termos de potenciais elétricos ϕ , o campo elétrico $\mathbf{E}$ pode ser escrito como

$$\mathbf{E} = -\nabla \phi. \quad (\text{A.4})$$

Substituindo (A.4) em (A.1), resulta na equação de Poisson

$$\nabla^2 \phi = -\frac{\hat{\rho}}{\varepsilon_r \varepsilon_0}. \quad (\text{A.5})$$

A equação de Poisson pode ser escrita em termos de condutividades. Se considerarmos a forma geral da lei de Ohm para um ponto no interior de um meio condutor,

$$\mathbf{J} = \sigma \mathbf{E} \quad (\text{A.6})$$

onde $\mathbf{J}$ é a densidade de corrente e σ é a condutividade. Como não há nenhuma fonte de corrente interna, pode-se escrever

$$\nabla \cdot \mathbf{J} = 0. \quad (\text{A.7})$$

Substituindo (A.4) em (A.6) e usando (A.7), tem-se a outra forma da equação de Poisson

$$\nabla \cdot (\sigma \nabla \phi) = 0. \quad (\text{A.8})$$

Esta é uma equação Elíptica na forma do divergente (EVANS, 2010). Neste trabalho, o meio é presumido isotrópico, de modo que a condutividade σ é um escalar. As condições de contorno (A.8) são dadas pelas correntes J na fronteira do domínio, assim

$$\sigma \frac{\partial \phi}{\partial \hat{n}} = J. \quad (\text{A.9})$$

A convenção adotada aqui é de valores de J positivos para correntes *deixando* o domínio. Condições de contorno que impõem valores para a primeira derivada da função como (A.9) são ditas *condições de contorno de Neumann*. Estas condições são pertinentes a esta particular formulação do problema direto, que tem como conhecidas as correntes na fronteira do domínio. Uma outra formulação possível do problema supõe conhecidos os potenciais na fronteira, levando a condições de contorno de Dirichlet.

Ainda sobre as condições de contorno, integrando-se (A.8) sobre o domínio Ω e aplicando-se o teorema de Gauss, tem-se

$$\oint_{\partial\Omega} \sigma \nabla \phi \cdot \hat{n} \, ds = 0. \quad (\text{A.10})$$

Substituindo-se (A.9) em (A.10), tem-se

$$\oint_{\partial\Omega} J \, ds = 0. \quad (\text{A.11})$$

Assim, (A.11) é uma condição necessária para que (A.8) tenha solução que satisfaça (A.9). Fisicamente, isso equivale a impor que a totalidade das correntes ingressando e deixando o domínio seja nula.

Para um domínio Ω genérico, soluções analíticas exatas para a equação (A.8) quando condições de contorno definidas por (A.9) não são conhecidas. Um modo de se obter soluções aproximadas para este problema é o Método dos Elementos Finitos (MEF) (CASTRO SOBRINHO, 2006; BATHE, 1995).

O MEF pode ser derivado da forma variacional do problema, obtida a partir da integração da equação de Poisson no domínio Ω^e de um elemento finito. Considerando o problema plano, a (A.8) assume a seguinte forma expandida

$$\frac{\partial}{\partial x} \left(\sigma \frac{\partial \phi}{\partial x} \right) + \frac{\partial}{\partial y} \left(\sigma \frac{\partial \phi}{\partial y} \right) = 0. \quad (\text{A.12})$$

A.1 Formulação Variacional

Multiplicando (A.12) por um função de ponderação $p(x, y)$ e integrando o domínio Ω^e do elemento obtém-se

$$\iint_{\Omega^e} p \left[\frac{\partial}{\partial x} \left(\sigma \frac{\partial \phi^e}{\partial x} \right) + \frac{\partial}{\partial y} \left(\sigma \frac{\partial \phi^e}{\partial y} \right) \right] dA = 0. \quad (\text{A.13})$$

Aplicando o teorema da derivada do produto de duas funções, expande-se a (A.13) em termos das derivadas parciais de primeira ordem,

$$\begin{aligned} \iint_{\Omega^e} \left[\frac{\partial p}{\partial x} \left(\sigma \frac{\partial \phi^e}{\partial x} \right) + \frac{\partial p}{\partial y} \left(\sigma \frac{\partial \phi^e}{\partial y} \right) \right] dA &= \\ \iint_{\Omega^e} \left[\frac{\partial}{\partial x} \left(p \sigma \frac{\partial \phi^e}{\partial x} \right) + \frac{\partial}{\partial y} \left(p \sigma \frac{\partial \phi^e}{\partial y} \right) \right] dA. \end{aligned} \quad (\text{A.14})$$

Pelo Teorema de Green, tem-se a seguinte expressão

$$\iint_{\Omega^e} \left[\frac{\partial p}{\partial x} \left(\sigma \frac{\partial \phi^e}{\partial x} \right) + \frac{\partial p}{\partial y} \left(\sigma \frac{\partial \phi^e}{\partial y} \right) \right] dA = \oint_{\underline{\Omega}^e} p \left(\sigma \frac{\partial \phi^e}{\partial x}, \sigma \frac{\partial \phi^e}{\partial y} \right) \cdot \hat{n} ds. \quad (\text{A.15})$$

Note que s é positivo no sentido anti-horário para os contornos externos e horário para os contornos internos. Note ainda que o vetor normal unitário $\hat{n}$ segue a mesma convenção.

A integral de linha de (A.15) representa o fluxo de corrente J^e através do contorno Ω^e do elemento. Este fluxo constitui a condição de contorno natural de (A.8). Assim, (A.15) pode ser escrita na forma

$$\iint_{\Omega^e} \left[\frac{\partial p}{\partial x} \left(\sigma \frac{\partial \phi^e}{\partial x} \right) + \frac{\partial p}{\partial y} \left(\sigma \frac{\partial \phi^e}{\partial y} \right) \right] dA = -J^e \quad (\text{A.16})$$

onde

$$J^e = - \oint_{\underline{\Omega}} p \left(\sigma \frac{\partial \phi^e}{\partial x}, \sigma \frac{\partial \phi^e}{\partial y} \right) \cdot \hat{n} ds \quad (\text{A.17})$$

sendo que o sinal negativo nas relações (A.16) e (A.17), entre J^e e a integral de linha, é devido ao fato do fluxo de corrente ter direção oposta ao gradiente do potencial elétrico. A (A.16) é a forma fraca ou forma variacional da equação de governo (A.8).

A.2 Funções de Interpolação

Devido à inexistência de soluções analíticas para o problema descrito, procura-se encontrar uma solução aproximada $\tilde{\phi}$ de (A.8), restrita a um espaço de funções gerado por uma *combinação linear* de funções-base que interpolam os valores de potencial ϕ_i^e nos pontos nodais.

Assim, seja $\Psi = [\psi_1, \psi_2, \dots, \psi_n]^T$ um vetor finito de n funções linearmente independentes tal que $\psi_i \in C(\mathbb{R}^2)$, ou seja, funções de $\mathbb{R}^2$ em $\mathbb{R}$ contínuas. Deste modo, o domínio $\mathcal{A}$ das funções exploradas pelo MEF é dado por

$$\mathcal{A} = \left\{ u : \mathbb{R}^2 \rightarrow \mathbb{R} \mid u = x^T \times \Psi, x \in \mathbb{R}^n \right\}. \quad (\text{A.18})$$

Assim, a função aproximada $\tilde{\phi}$ pode ser escrita como

$$\tilde{\phi} = \Phi^T \Psi \quad (\text{A.19})$$

onde $\Phi = [\phi_1, \phi_2, \dots, \phi_n]^T$ é o vetor de coeficientes que define $\tilde{\phi}$. As funções-base ψ_i são também chamadas *funções de forma*.

A.3 Montagem do Sistema de Equações Lineares do Modelo

Para obter um sistema de equações cujas incógnitas são os valores de potencial ϕ_i^e nos nós do elemento, é necessário atribuir valores às funções de ponderação $p(x, y)$ utilizadas em (A.13). Como $p(x, y)$ pode assumir qualquer valor, adota-se o uso de funções de interpolação ψ_j , para $j = 1, \dots, m$. O índice m é definido pelo elemento geométrico empregado. Utilizando uma malha de elementos triangulares isoparamétricos, tem-se $m = 3$. Substituindo em (A.16) e (A.17), obtém-se

$$\iint_{\Omega^e} \sigma \left[\frac{\partial \psi_j}{\partial x} \sum_{i=1}^3 \left(\frac{\partial \psi_i}{\partial x} \phi_i^e \right) + \frac{\partial \psi_j}{\partial y} \sum_{i=1}^3 \left(\frac{\partial \psi_i}{\partial y} \phi_i^e \right) \right] dA = -J_j^e \quad (\text{A.20})$$

onde,

$$J_j^e = - \oint_{\underline{\Omega}^e} \psi_j \sigma \left[\left(\sum_{i=1}^3 \frac{\partial \psi_i}{\partial x} \phi_i^e \right), \left(\sum_{i=1}^3 \frac{\partial \psi_i}{\partial y} \phi_i^e \right) \right] \cdot \hat{n} ds. \quad (\text{A.21})$$

Explicitando (A.20) e (A.21), para os valores de j , na forma matricial $\mathbf{K}^e \phi^e = \mathbf{J}^e$, tem-se

$$\begin{bmatrix} K_{1,1}^e & K_{1,2}^e & K_{1,3}^e \\ K_{2,1}^e & K_{2,2}^e & K_{2,3}^e \\ K_{3,1}^e & K_{3,2}^e & K_{3,3}^e \end{bmatrix} \begin{bmatrix} \phi_1^e \\ \phi_2^e \\ \phi_3^e \end{bmatrix} = \begin{bmatrix} -J_1^e \\ -J_2^e \\ -J_3^e \end{bmatrix} \quad (\text{A.22})$$

onde

$$K_{i,j}^e = \iint_{\Omega^e} \sigma \left(\frac{\partial \psi_i}{\partial x} \frac{\partial \psi_j}{\partial x} + \frac{\partial \psi_i}{\partial y} \frac{\partial \psi_j}{\partial y} \right) dA. \quad (\text{A.23})$$

Note que $K_{i,j}^e = K_{j,i}^e$, o que torna a matriz do sistema $\mathbf{K}^e \phi^e = \mathbf{J}^e$, expressa em (A.22), simétrica. Aplica-se então a formulação desenvolvida para um elemento isolado, sobre o domínio Ω^e , discretizando a região contínua deste domínio com o uso de uma malha de elementos finitos.

Para montar o sistema de equações, adicionam-se as constantes das matrizes de cada um dos elementos às constantes da matriz modelo, seguindo as correspondências dadas

por funções de conectividade (CASTRO SOBRINHO, 2006). Tais funções armazenam a informação dos nós globais, ou seja, a posição em relação à malha como um todo, associados aos nós locais, posição relativa ao elemento em questão. O mesmo método é aplicado ao vetor de fluxos $\mathbf{J}$.

Observa-se que tanto a matriz $\mathbf{K}$ quanto o vetor $\mathbf{J}$ possuirão valores nulos para as variáveis ϕ_i que não tenham correspondência com os nós locais do elemento Ω^e , resultando em uma matriz $\mathbf{K}$ esparsa. Nota-se também que esta é simétrica, devido à simetria das matrizes dos elementos. Esta propriedade pode ser utilizada para reduzir o trabalho de montagem do sistema, uma vez que deve-se calcular apenas as constantes acima ou abaixo da diagonal principal e os da própria diagonal principal.

A.4 Cálculo Analítico das Constantes do Sistema

As constantes do sistema (A.22) são obtidas pela integração dos elementos da matriz $\mathbf{K}$ e do vetor $\mathbf{J}$. Para tal, utiliza-se o domínio das coordenadas naturais, o que torna necessária a transformação de coordenadas do sistema global (x, y) para o sistema natural de coordenadas $-1 \leq r, s \leq 1$.

Esta mudança é feita pela definição da área diferencial dA , em coordenadas naturais. Note que no sistema das coordenadas globais (x, y) , a área diferencial é dada por $dA = dx dy = |(dx\hat{x}) \times (dy\hat{y})| = |\hat{x} \times \hat{y}| dx dy$, onde $\hat{x}$ e $\hat{y}$ são vetores unitários do sistema de coordenadas global. O mesmo dA pode ser definido para as coordenadas naturais, ou seja

$$dA = |(dr\hat{r}) \times (ds\hat{s})| = |\hat{r} \times \hat{s}| dr ds \quad (\text{A.24})$$

em que os vetores unitários são obtidos pela variação de um vetor qualquer $\vec{v} = (x, y)$, nas direções das coordenadas naturais r e s ,

$$\hat{r} = \frac{\partial \vec{v}}{\partial r} = \left(\frac{\partial x}{\partial r}, \frac{\partial y}{\partial r} \right) \quad \hat{s} = \frac{\partial \vec{v}}{\partial s} = \left(\frac{\partial x}{\partial s}, \frac{\partial y}{\partial s} \right) \quad (\text{A.25})$$

Substituindo (A.25) em (A.24), obtém-se

$$dA = dx dy = \det \begin{bmatrix} \frac{\partial x}{\partial r} & \frac{\partial y}{\partial r} \\ \frac{\partial x}{\partial s} & \frac{\partial y}{\partial s} \end{bmatrix} dr ds = (\det \mathbf{Q}) dr ds$$

onde $\mathbf{Q}$ é a matriz Jacobiana da transformação de coordenadas. Deste modo, chega-se na forma em que serão calculados os coeficientes da matriz $\mathbf{K}$ definidos em (A.23),

$$K_{i,j}^e = \int_{-1}^{+1} \int_{-1}^{+1} \sigma \left(\frac{\partial \psi_i}{\partial x} \frac{\partial \psi_j}{\partial x} + \frac{\partial \psi_i}{\partial y} \frac{\partial \psi_j}{\partial y} \right) \det \mathbf{Q} dr ds \quad (\text{A.26})$$

Para obter as derivadas das funções de interpolação ψ_i no sistema de coordenadas (r, s) , são necessárias transformações de variáveis de derivação, ou seja

$$\frac{\partial \psi_i}{\partial r} = \frac{\partial \psi_i}{\partial x} \frac{\partial x}{\partial r} + \frac{\partial \psi_i}{\partial y} \frac{\partial y}{\partial r}$$

$$\frac{\partial \psi_i}{\partial s} = \frac{\partial \psi_i}{\partial x} \frac{\partial x}{\partial s} + \frac{\partial \psi_i}{\partial y} \frac{\partial y}{\partial s}.$$

Na forma matricial, tem-se

$$\begin{bmatrix} \frac{\partial \psi_i}{\partial r} \\ \frac{\partial \psi_i}{\partial s} \end{bmatrix} = \begin{bmatrix} \frac{\partial x}{\partial r} & \frac{\partial y}{\partial r} \\ \frac{\partial x}{\partial s} & \frac{\partial y}{\partial s} \end{bmatrix} \begin{bmatrix} \frac{\partial \psi_i}{\partial x} \\ \frac{\partial \psi_i}{\partial y} \end{bmatrix} = \mathbf{Q} \begin{bmatrix} \frac{\partial \psi_i}{\partial x} \\ \frac{\partial \psi_i}{\partial y} \end{bmatrix}$$

$$\begin{bmatrix} \frac{\partial \psi_i}{\partial x} \\ \frac{\partial \psi_i}{\partial y} \end{bmatrix} = \mathbf{Q}^{-1} \begin{bmatrix} \frac{\partial \psi_i}{\partial r} \\ \frac{\partial \psi_i}{\partial s} \end{bmatrix}$$

APÊNDICE B – Avaliação Incompleta da Função Objetivo

A avaliação da função objetivo é responsável pela maior parte do custo computacional do recozimento simulado. Portanto, métodos para reduzir seu custo são de extremo interesse, e uma forma possível é através da avaliação incompleta da função objetivo, como introduzida por Lakshmanan e Derin ([LAKSHMANAN, 1989](#)). Neste trabalho, é mostrado o comportamento do recozimento simulado na presença da avaliação incompleta da função objetivo. A distribuição exata da função objetivo é presumida como desconhecida, mas os limites para seus valores são estimados. Como mostrado por Martins et al. ([MARTINS; TSUZUKI, 2011](#)), há uma relação entre estes limites e a probabilidade da convergência desviar da solução exata. Utilizando-se essas relações e impondo-se limites para essas probabilidades, um critério de parada para o cálculo iterativo da função objetivo pode ser obtido.

Meurant ([MEURANT, 2005](#)) propôs um método para estimar limites superiores para a norma l_2 do erro dos Gradientes Conjugados em cada iteração. Estes limites podem ser utilizados para estimar os limites do erro para a função objetivo (4.1) quando soluções exatas para ϕ_c determinadas por MEF são substituídas por soluções parciais $\tilde{\phi}_c$ obtidas pela parada dos Gradientes Conjugados antes da convergência final. Isto é realizado assumindo que todo o erro está concentrado nos nós dos eletrodos.

Então, se $d_i = |\phi_m^i - \phi_c^i|$ é a componente da função objetivo para um dado padrão de corrente, então

$$\tilde{d}_i = |\phi_m^i - \tilde{\phi}_c^i| \quad (\text{B.1})$$

é a estimativa obtida com a solução parcial $\tilde{\phi}_c$ e

$$\epsilon^i \geq |\phi_c^i - \tilde{\phi}_c^i| \quad (\text{B.2})$$

é o limite superior para o erro dos Gradientes Conjugados, tem-se que

$$\begin{aligned} D^i \leq d_{\max}^i &= \tilde{d}^i + \epsilon^i \\ D^i \geq d_{\min}^i &= \max(0, \tilde{d}^i - \epsilon^i) \end{aligned} \quad (\text{B.3})$$

e, portanto,

$$\begin{aligned} \tilde{E} &= \sqrt{\Sigma(\tilde{d}^i)^2} \\ E \leq E_{\max} &= \sqrt{\Sigma(d_{\max}^i)^2} \\ E \geq E_{\min} &= \sqrt{\Sigma(d_{\min}^i)^2}. \end{aligned} \quad (\text{B.4})$$

Os limites superior e inferior para E convergem para o valor exato, pois o número de iterações dos múltiplos algoritmos de Gradientes Conjugados aumentam. Como o recozimento simulado é sensível apenas a variações relativas da função objetivo estes limites precisam ser convertidos para limites de variação de E . Se $\Delta E = E^j - E^i$ é a variação da função objetivo quando o recozimento simulado movimenta de uma solução x^j para outra x^{j+1} então, por aritmética intervalar,

$$\begin{aligned}\Delta \tilde{E} &= \tilde{E}^j - \tilde{E}^{j+1} \\ \Delta E \leq \Delta E_{\max} &= E_{\max}^j - E_{\min}^{j+1} \\ \Delta E \geq \Delta E_{\min} &= E_{\min}^j - E_{\max}^{j+1}.\end{aligned}\tag{B.5}$$

Cujos limites para ΔE convergem para um valor exato, mas desta vez, é necessário aumentar o número de iterações para ambas as avaliações de x^j e x^{j+1} . Assim, é possível observar que um recozimento simulado que utiliza soluções parciais do MEF $\tilde{\phi}_c$ ao invés de soluções exatas ϕ_c pode possuir probabilidades limitadas de divergir de um recozimento simulado que utiliza as soluções exatas. Entretanto, ao impor P_{err} como limite superior para a probabilidade do recozimento simulado decidir incorretamente (rejeitar uma solução quando ele deveria aceitá-la ou aceitá-la quando ele deveria rejeitá-la), é possível criar condições para os limites expressos em (B.5), definido por

$$e^{-\frac{\Delta E_{\max}}{kt}} \geq \begin{cases} 1 - P_{err} & \text{se } \Delta \tilde{E} \leq 0, \\ e^{-\frac{\Delta \tilde{E}}{kt}} - P_{err} & \text{se } \Delta \tilde{E} > 0 \end{cases}\tag{B.6}$$

$$e^{-\frac{\Delta E_{\min}}{kt}} \leq \min(1, e^{-\frac{\Delta \tilde{E}}{kt}} + P_{err}).\tag{B.7}$$

As condições fornecidas por (B.6) e (B.7) não podem ser diretamente traduzidas em critérios de parada para os Gradientes Conjugados, como $\Delta E_{\max}$ e $\Delta E_{\min}$ são soluções parciais de dois problemas de MEF, e geralmente, eles não podem ser obtidos pelo estreitamento dos limites de apenas um deles. Assim, toda solução precisa ter armazenado todo o processo de avaliação da função objetivo e assim será possível continuá-lo a partir do ponto em que ele foi parado.

De fato, é computacionalmente mais eficiente estreitar os limites sobre a avaliação anterior do recozimento simulado que na atual. Como o algoritmo dos Gradientes Conjugados possui convergência geométrica assintótica e presumindo que ambas as soluções do recozimento simulado, a anterior e a atual, possuem taxas de convergência similares (uma suposição razoável é considerar que a matriz $\mathbf{K}$ varia muito pouco de uma iteração para a outra e um pré-condicionador equaliza ainda mais a taxa de convergência), uma heurística simples para selecionar qual solução deve ser melhor avaliada é escolher a que possuir limites mais apertados.

Da mesma maneira, uma análise muito similar ocorre na determinação de qual modelo de MEF (correspondente a um padrão de corrente) deve ser melhor avaliado, uma

Figura 72 – Domínio de simulação.

heurística simples é selecionar aquele com maior valor de $d_i + \epsilon_i$.

B.1 Resultados Obtidos pela Implementação do Método

Para avaliar a viabilidade do recozimento simulado com avaliação parcial aplicado ao problema da TIE, uma implementação simples foi desenvolvida utilizando um domínio simulado (vide Fig. 72). O domínio simulado é quadrado, medindo 16×16 unidades, composto de 76 elementos triangulares com 320 nós, dos quais 32 são eletrodos. A condutância elétrica foi dividida em 64 quadrados 2×2 (observe que a discretização do potencial e a discretização da condutância não são as mesmas). A condutância em cada elemento é definida no intervalo $[1, 2]$.

Os eletrodos são modelados como retângulos 2×1 , a maior aresta está em contato com o domínio. A aresta oposta é modelada como estando em contato com um elemento de condutância infinita e consequentemente o potencial ao longo da aresta é constante (a forma triangular na Fig. 72 pode produzir interpretações incorretas, esta forma apenas reflete o fato de que a maior aresta foi colapsada em um único nó de potencial no modelo do MEF). O primeiro eletrodo (eletrodo superior na coluna à esquerda) foi definido como referência. Foram aplicados 31 padrões de corrente, correspondentes a uma corrente unitária entrando em cada eletrodo, exceto na referência, e saindo pelo eletrodo de referência.

Os problemas de MEF foram solucionados utilizando os Gradientes Conjugados com pré-condicionamento, o pré-condicionador foi a decomposição de Cholesky incompleta (KERSHAW, 1978). Como sugerido por Meurant (MEURANT, 2005), o pré-condicionador com norma l_∞ foi utilizado para estimar o erro de norma l_2 . Os 64 parâmetros de condutividade foram gerados pelo recozimento simulado. Foi utilizada a heurística de vizinhança proposta por Martins e Tsuzuki (MARTINS; TSUZUKI, 2008b), modificando apenas um parâmetro de condutividade por vez em cada iteração e reduzindo a probabilidade de gerar modificações que levam a soluções rejeitadas. A probabilidade de divergência P_{err} foi definida arbitrariamente como $1/100$.

Figura 73 – Comportamento da convergência do problema do tabuleiro de xadrez.

Figura 74 – Energia por temperatura determinada para o problema do tabuleiro de xadrez (as barras de erro representam o desvio padrão da energia para a temperatura).

B.2 Problema do Tabuleiro de Xadrez

Foi proposto outro problema complexo: um tabuleiro de xadrez com padrões de condutividade 4×4 , alternando alta condutância 2 com baixa condutância 1.

O recozimento simulado reconstruiu precisamente a distribuição de condutividade. Examinando resultados intermediários, exibidos na Fig. 73.(a)-(f) e na Fig. 74, é possível observar que a distribuição de impedância foi reconstruída do exterior para o interior do domínio. Assim, é razoável presumir que a função objetivo é mais sensível a modificações

nos elementos externos. Estes resultados sugerem que para o recozimento simulado, a avaliação parcial da função objetivo pode ser tão efetiva quanto a avaliação completa da função objetivo. A Fig. 75 exibe o número médio de iterações dos Gradientes Conjugados utilizado pelo recozimento simulado em cada temperatura.

Considerando que o problema possui 320 nós, é surpreendente que o recozimento simulado obteve estes resultados com menos de 20 iterações dos Gradientes Conjugados em média. Mais interessante ainda é a evolução no número de iterações de acordo com o progresso da otimização. Em altas temperaturas, os valores elevados de kt e $\Delta\tilde{E}$ levam a condições relativamente soltas em (B.6) e (B.7), atingível com poucas iterações dos Gradientes Conjugados. De acordo com a diminuição da temperatura, valores menores de kt e $\Delta\tilde{E}$ (à medida em que as soluções avaliadas se aproximam) impõem condições mais apertadas para (B.6) e (B.7), necessitando de mais iterações dos Gradientes Conjugados. Pode ser surpreendente que à medida que o recozimento simulado converge para a solução final o número de iterações dos Gradientes Conjugados reduza novamente. Isto pode ser explicado por dois fatores. Primeiro, o chute inicial para os Gradientes Conjugados é obtido a partir da solução obtida anteriormente pelo recozimento simulado. Segundo, a heurística de vizinhança adaptativa reduz dramaticamente as modificações da distribuição de condutividade à medida que o recozimento simulado converge para o ótimo global (MARTINS; TSUZUKI, 2008b). Combinando estes dois fatores, é possível observar que durante a etapa de convergência final do recozimento simulado, a solução anterior do MEF é um chute inicial muito bom para compensar as condições extremas dos limites de ΔE .

Em uma das observações feitas por Martins et al. (MARTINS; TSUZUKI, 2011), comentou-se que, durante a convergência do problema inverso, não foram necessárias mais do que 32 iterações, em média, para se avaliar com confiança a solução objetivo, o que equivale a cerca de 3% do total de iterações. Assim, neste trabalho o número de iterações de Gradientes Conjugados será limitado em cerca de 5% do total.

Figura 75 – Iterações de Gradientes Conjugados (vertical) x temperatura (horizontal) para o problema do tabuleiro de xadrez.