UNIVERSIDADE FEDERAL DA BAHIA
INSTITUTO DE GEOCIÊNCIAS
CURSO DE GRADUAÇÃO EM GEOFÍSICA

GEO213 – TRABALHO DE GRADUAÇÃO

MODELAGEM SÍSMICA E MIGRAÇÃO REVERSA NO TEMPO: UMA IMPLEMENTAÇÃO USANDO PLACAS GRÁFICAS (GPU)

VICTOR KOEHNE RAMALHO

SALVADOR – BAHIA
AGOSTO – 2014

Modelagem sísmica e migração reversa no tempo: uma implementação usando placas gráficas (GPU)

por

Victor Koehne Ramalho

Orientador: Prof. Dr. Reynam da Cruz Pestana

GEO213 – TRABALHO DE GRADUAÇÃO

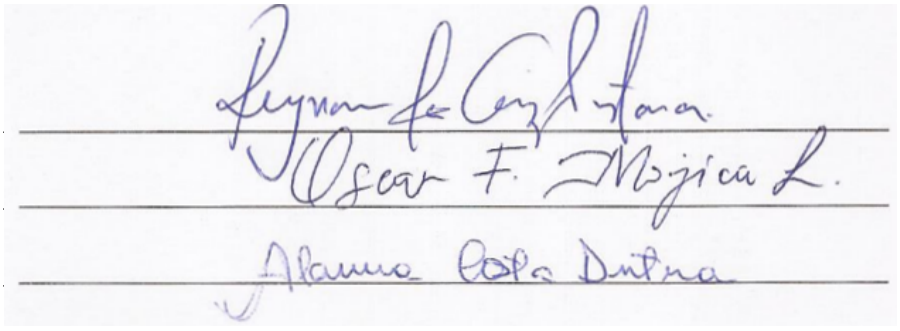
Departamento de Geofísica

do

Instituto de Geociências

da

Universidade Federal da Bahia



Comissão Examinadora

Reynam da Cruz Pestana

Oscar Mojica Ladino

Alanna Costa Dutra

Data da aprovação: 05/08/2014

ÍNDICE

ÍNDICE DE FIGURAS

RESUMO

A migração reversa no tempo (RTM - *Reverse Time Migration*) é um dos métodos de imageamento mais precisos na migração de dados sísmicos de modelos geológicos de grande complexidade, como variações laterais de velocidade, falhas, dobras e diapirismo. Entretanto, sua principal restrição é a alta demanda compucional, tanto em tempo de execução quanto em espaço de armazenamento de informações temporárias. Portanto, sua aplicação implica no desenvolvimento de estratégias que os tornem viáveis na prática. Neste trabalho de graduação a RTM é implementada em algoritmos desenvolvidos para computação paralela, utilizando unidades de processamento gráfico (GPUs); associado a isto, é aplicado o método de expansão rápida (REM) para realizar a extrapolação do campo de onda de forma mais precisa.

Foram realizadas a modelagem e migração sísmica de dois modelos geológicos complexos: Marmousi, que apresenta falhamentos, dobras e cunhas salinas; modelo Falhas, que também apresenta grande quantidade de falhamentos e diapirismo. Foram migrados em paralelo os dados públicos de tiro do modelo de velocidades Marmousi, modelados por algoritmos de diferenças finitas pelo *Institute Français du Pétrole*, com ganho considerável de tempo comparado à migração em código serial em trabalhos anteriores. Também foi realizada a modelagem destes tiros utilizando o mesmo campo de velocidades e com geometria idêntica, aplicando o método de expansão rápida, e executada migração reversa no tempo em GPU, cuja imagem resultante apresentou qualidade similar à migração do dado anterior.

ABSTRACT

Among the seismic imaging methods, Reverse Time Migration (RTM) is one of the most precise when migrating seismic data of geological models of great complexity, such as strong lateral velocity variations, faults, folds and diapirs. However the main restriction is its high computational demand, both in running time and in storage of temporary data. Therefore, the application of RTM implies on developing strategies to make it feasible on practice. In this undergraduate work the RTM algorithms are implemented using parallel computation, more specifically on a graphics processing unit (GPU); associated to that it is also applied the rapid expansion method (REM) to compute the wavefield extrapolation more accurately.

Seismic modelling and reverse time migration were carried out for two complex geological models: the Marmousi model, containing faults, folds and salt wedges; the "Faults" model, which contains a high number of faults and also diapirism. The first migration was applied to the public Marmousi (*Institute Français du Pétrole*) shot gathers, modelled with a finite differences algorithm, with a considerable computational gain when compared to the reverse time migration of this same data carried out in previous works in only one processor, and with similar quality on both results. Secondly it was carried out a modelling of these shots using the same velocity field and with identical geometry, implemented with the rapid expansion method, and this modelling data migrated with RTM GPU algorithm. Both the final migrated sections showed similar quality, validating the use of REM and GPU RTM on the parallelized algorithm.

INTRODUÇÃO

Na exploração de hidrocarbonetos, o método sísmico é de fundamental importância na procura e delineação de reservatórios de óleo e gás. Dentro do método sísmico, a sísmica de reflexão é a principal ferramenta utilizada atualmente para este propósito. Na exploração por sísmica de reflexão, existem três estágios principais: aquisição de dados, processamento e interpretação. O foco deste trabalho inclui-se no segundo estágio, processamento de dados sísmicos, mais especificamente a migração pré-empilhamento em profundidade.

A aquisição de dados sísmicos de reflexão requer uma fonte sísmica de energia, que é propagada como onda através da subsuperfície. Ao encontrar uma variação no meio, parte da energia da onda é refletida e parte é transmitida. As reflexões da onda são registradas em receptores, que medem o tempo de percurso e a amplitude destas chegadas. Desta forma, cada receptor - ou geofone - ao final da aquisição, contém um registro de tempo e amplitudes, conhecido como traço sísmico. O conjunto de traços gravados pelos geofones para uma determinada fonte é chamado família de tiro comum. O arranjo geofones-fonte é então deslocado e um novo tiro é disparado, gerando novos registros da próxima família de tiro comum. O conjunto de todos os registros de todos os tiros constitui o dado sísmico.

Esse dado sísmico obtido na fase de aquisição passará por uma série de etapas de processamento. A sequência clássica de processamento, consolidada na indústria e no meio acadêmico, consiste em três processos fundamentais: deconvolução, empilhamento e migração (Yilmaz, 2001). O objetivo da deconvolução é comprimir a assinatura da fonte (wavelet), atenuar reverberações e múltiplas de curto período, desta forma elevando consideravelmente a resolução temporal dos registros. O empilhamento consiste em reorganizar as famílias de tiro comum em famílias de ponto médio comum (CMP), somando os traços internos a cada família e ao final organizando cada traço empilhado em uma única seção sísmica, a seção empilhada; este processo aumenta drasticamente a razão sinal/ruído do dado sísmico.

O aumento de resolução da técnica de empilhamento CMP, porém, considera que os refletores presentes no modelo são planos, paralelos e horizontais. Esta suposição normalmente não é obedecida, pois comumente a subsuperfície apresenta estruturas complexas como falhas, dobras, diápiros e refletores com grande inclinação. Desta forma, a posição dos refletores no dado pode ser diferente da posição real na subsuperfície. Na abordagem do processamento clássica, cabe à migração pós-empilhamento corrigir estes efeitos.

A migração sísmica move reflexões inclinadas para suas verdadeiras posições na subsuperfície e colapsa difrações, aumentando assim a resolução espacial e produzindo uma

imagem sísmica mais fidedigna da supsuperfície. O objetivo da migração pós-empilhamento é fazer com que a seção empilhada se assemelhe à seção geológica em profundidade (Yilmaz, 2001). Todavia quando a subsuperfície apresenta feições de grande complexidade, como grandes contrastes laterais de velocidade, falhas, dobras e diápiros, estas feições não são bem imageadas, e é necessário que se realize a migração pré-empilhamento em profundidade. Isto se deve ao fato da migração pré-empilhamento em profundidade ser mais genérica, por não partir de premissas sobre o modelo geológico (como a suposição de que os refletores são plano paralelos horizontais, por exemplo).

O foco deste trabalho é imagear, através da migração pré-empilhamento em profundidade, modelos gelógicos de grande complexidade, cuja imagem final gerada pelo fluxograma clássico de processamento pós-empilhamento não consegue delinear as feições complexas da subsuperfície. Mais especificamente, a técnica de migração pré-empilhamento utilizada neste trabalho é a *Reverse Time Migration* (RTM), ou migração reversa no tempo (Baysal et al., 1983a). Este método permite migrar com precisão refletores com qualquer inclinação, em modelos com variação lateral de velocidade, e utiliza os campos de onda ascendente e descendente ao gerar a imagem sísmica. A principal razão deste método não ter sido aplicado usualmente na indústria, ao longo dos anos, se deve ao seu elevado custo computacional, inviável para a tecnologia da época. Durante muitos anos, a RTM foi alvo apenas de estudos teóricos. Atualmente, o poder de cálculo dos processadores aumentou vertiginosamente, que aliado à possibilidade de implementar programas em paralelo - seja em diversos processadores, ou em unidades de placa gráfica (GPU) - permitiu que a RTM fosse aplicada e seus resultados analisados, como neste trabalho.

Na migração reversa no tempo são utilizados algoritmos de modelagem sísmica, onde as derivadas espaciais são calculadas por esquemas de diferenças finitas ou pelo método de Fourier. A integração no tempo é geralmente feita usando-se diferenças finitas de baixa ordem. Quando as derivadas espaciais são calculadas por esquemas de alta ordem, existe um desbalanceamento que, normalmente, requer que o passo de avanço no tempo seja muito pequeno para evitar dispersão numérica. Como resultado disso, se pode usar o método REM para intervalos maiores de amostragem no tempo (se necessário), e a instabilidade numérica é evitada quando a aproximação em série de Chebyshev é aplicada usando-se o número de termos apropriado (Pestana e Stoffa, 2010).

Quanto ao problema computacional, existem duas formas de reduzir o tempo de execução de um programa serial sem realizar uma refatoração do código. A primeira é aumentar o número de operações por segundo, usando um processador mais rápido. Já a segunda solução consiste em computar as partes seriais do código simultaneamente em paralelo. Assim, N processadores podem ser usados ao mesmo tempo, reduzindo o tempo de processamento, sem exigir processadores mais rápidos.

Uma alternativa mais recente, que será focada neste trabalho, é o uso de placas gráficas

(GPU). As GPUs são originalmente unidades de processamento gráfico, por isso tem sua estrutura otimizada para a realização de operações de ponto flutuante. Essa característica das GPUs passou a ser explorada por outras áreas da ciência, a fim de melhorar o desempenho de programas que consistem basicamente de operações de ponto flutuante, como é o caso da modelagem sísmica e da migração reversa no tempo.

Neste trabalho o principal objetivo é implementar a RTM com uma melhor precisão na integração do tempo, através do REM, e usar algoritmos implementados em GPUs. Desta forma, a combinação do REM, que é o método considerado mais preciso para integração temporal, com uma implementação em GPUs, que é uma das tecnologias mais eficientes disponíveis, possibilita resultados de modelagem e migração mais precisos e também a um menor custo computacional quando comparado a migração reversa no tempo com REM executada em um único processador.

O trabalho está dividido da seguinte forma: no Capítulo 1 são apresentados os conceitos de modelagem sísmica e RTM. São também apresentadas as equações fundamentais para o REM, e os fluxogramas da modelagem e RTM. No Capítulo 2 é discutida a metodologia utilizada na implementação da modelagem e RTM. No Capítulo 3 é apresentada a GPU, seus usos, aplicações à sísmica e uma análise das suas vantagens comparado a um processador serial. No Capítulo 4 são apresentados resultados de modelagem e RTM para dois modelos geológicos complexos. O Capítulo 5 destina-se às conclusões.

CAPÍTULO 1

Apresentação Teórica

Dado que o foco deste trabalho é avaliar a eficácia da implementação da RTM, com utilização do REM, em algoritmos paralelizados, o entendimento da modelagem sísmica é importante por dois aspectos: mostrar que a forma de modelagem implementada neste trabalho tem qualidade similar aos dados modelados de tiros de domínio público (mais especificamente, dados de tiro gerados por algoritmos de diferenças finitas para o modelo Marmousi, disponibilizados pelo *Institute Français du Pétrole*), através da comparação das imagens migradas do dado público e do dado modelado em GPU; e estabelecer uma das bases teóricas para o entendimento da migração reversa no tempo, visto que resultados de modelagem a partir do campo de velocidades são utilizados a cada iteração da RTM para construir a imagem sísmica final.

Desta forma, neste capítulo são apresentados os conceitos teóricos fundamentais ao entendimento da modelagem sísmica 2D: equação da onda acústica, aproximação de derivadas espaciais por operadores de diferenças finitas e extrapolação temporal pelo método de expansão rápida, e suas vantagens.

Estabelecido o conceito de modelagem sísmica, é introduzida a fundamentação teórica da migração reversa no tempo pré-empilhamento, cujo conceito de modelagem se faz imprescindível à compreensão da condição de imagem da RTM. As etapas fundamentais do processo são brevemente descritas, e outros conceitos fundamentais ao seu entendimento são introduzidos: o modelo dos refletores explosivos e a condição de imagem.

1.1 Modelagem Sísmica

A modelagem sísmica simula a propagação de uma onda sísmica em subsuperfície. Neste trabalho a subsuperfície é formulada como uma malha 2D, onde cada ponto (x, z) desta malha assume um valor de pressão $P(x, z, t)$ e velocidade sísmica $v(x, z)$, correspondente à fatia do campo de velocidades no qual se quer realizar a modelagem do campo de pressões $P(x, z, t)$. Inicialmente (tempo zero), $P(x, z, t) = 0$ em toda a malha, exceto na posição na qual se introduz o pulso sísmico $s(t)$; em outras palavras, sendo (x_s, z_s) a posição espacial do pulso sísmico na malha, $P(x_s, z_s, t) = s(t)$. O pulso ideal seria uma delta de Dirac, a

qual possui amplitude apenas quando $t = 0s$. No caso de modelar um pulso real, este tem uma pequena duração temporal, por exemplo, se for usado um pulso (ou wavelet) do tipo Ricker como aproximação da fonte, com frequência máxima $35Hz$, $t_{pulso} = 1/35 \approx 28ms$, o que significa que o tempo para que toda a forma do pulso seja inserida no modelo é de $28ms$. Isto significa que durante os primeiros $28ms$ de modelagem, o pulso é tanto inserido no modelo quanto propagado por ele; após $28ms$ há apenas propagação.

A informação proveniente da modelagem acústica realizada desta maneira pode ser salva de diversas formas, dependendo do foco de estudo ou do método que se quer implementar. Na modelagem de um tiro sísmico, por exemplo, só são salvas informações do campo na superfície, ou seja, a matriz $P(x, z = 0, t)$. No caso de um Perfil Sísmico Vertical (*Vertical Seismic Profile* ou VSP), seria salva a matriz $P(x = x_{VSP}, z, t)$. Uma terceira forma é armazenar todo o campo de pressão em todos os tempos, ou seja, a matriz 3D completa $P(x, z, t)$, denominados "instantâneos" da propagação da onda ou "snapshots".

A migração reversa no tempo pré-empilhamento é feita tiro a tiro, e a cada tiro é realizada uma modelagem sísmica na qual são salvos todos os seus snapshots $P(x, z, t)$ correspondentes à fatia do modelo que se está imageando. O imageamento sísmico via RTM utiliza snapshots da modelagem sísmica salvos exatamente da forma aqui descrita. As demais etapas da migração reversa no tempo, e sua relação com os snapshots da modelagem sísmica são detalhadas na seção final deste capítulo.

A modelagem de tiros tem elevado custo de processamento, no caso dos modelos 2D testados neste trabalho. O tempo de modelagem dos tiros do dado Marmousi (com uso do REM), por exemplo, com um campo de velocidade de dimensão 369×375 amostras espaciais, e 725 amostras temporais por traço, leva um tempo de aproximadamente $3h36min$ em um único processador, de acordo com resultados obtidos em dos Santos (2010). Na implementação deste trabalho, a modelagem deste modelo com as mesmas amostragens espaciais e o mesmo número de amostras temporais por traço levou um tempo aproximado de $3min54s$. Analogamente, a migração reversa no tempo dos 240 tiros do dado público Marmousi disponibilizado pelo *Institute Français du Pétrole* leva um tempo aproximado de $7h15min$ (com uso do REM), em um único processador (dos Santos, 2010); no algoritmo implementado em GPU (também com uso do REM), usando dimensões e amostragens similares, o tempo de processamento foi de $5min54s$. Esta análise permite concluir que o uso de algoritmos de modelagem paralelizados em GPU é de grande utilidade, pois permite que mais testes sejam realizados e mais resultados sejam gerados, a um menor custo de tempo.

1.1.1 Equação da Onda Acústica

A propagação de onda realizada na modelagem é feita a partir da extrapolação do campo de pressões $P(x, z, t)$, através da equação 2D da onda acústica:

$$\frac{1}{c(x,z)^2}\frac{\partial^2 P(x,z,t)}{\partial t^2} = \nabla^2 P(x,z,t) + S(x,z,t) \tag{1.1}$$

onde ∇^2 é o operador laplaciano, dado por $\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial z^2}$, $P(x,z,t)$ é o campo de onda, $c(x,z)$ é a velocidade de propagação, x e z são as coordenadas espaciais do modelo 2-D e t é o tempo. $S(x,z,t)$ é o termo fonte discutido anteriormente; para $(x,z) = (x_s, z_s)$, $S(x_s, z_s, t) = s(t)$, e nos demais casos $S(x,z,t) = 0$.

A modelagem consiste em prever o campo de pressão no futuro com base nas configurações atual e passada, ou seja, calcular $P(x,z,t+\Delta t)$ com base em $P(x,z,t)$ e $P(x,z,t-\Delta t)$. Para isto, aproxima-se a derivada temporal de (1.1) por uma diferença finita centrada (Araújo, 2009):

$$\frac{\partial^2 P(x,z,t)}{\partial t^2} \approx \frac{P(x,z,t+\Delta t) - 2P(x,z,t) + P(x,z,t-\Delta t)}{\Delta t^2} \tag{1.2}$$

de forma que o campo de onda no tempo $(t+\Delta t)$ pode ser aproximado pela equação abaixo,

$$P(x,z,t+\Delta t) \approx 2P(x,z,t) - P(x,z,t-\Delta t) + \Delta t^2 c^2 \nabla^2 P(x,z,t) + S(x,z,t) \tag{1.3}$$

Esta equação é aplicada a cada ponto da malha na qual se está modelando, a cada iteração temporal sendo responsável pela propagação do campo de onda. O operador ∇^2 pode ser aproximado por diversos métodos; neste trabalho é utilizada a aproximação por diferenças finitas de quarta ordem (ver seção 1.2.1).

A aproximação da derivada temporal segunda de $P(x,z,t)$, feita por diferenças finitas de segunda ordem na equação (1.2), pode ser feita de outra forma, que apresenta melhor acurácia na propagação do campo de onda em qualquer amostragem no tempo: o Método de Expansão Rápida, ou Rapid Expansion Method (REM) Pestana e Stoffa, 2010. O REM será melhor apresentado na seção final do Capítulo.

A Figura 1.1(a) mostra um modelo de velocidades com duas camadas, com velocidades de propagação de $1000m/s$ e $2000m/s$. Nas Figuras 1.1(b), 1.1(c) e 1.1(d) é possível ver snapshots de uma modelagem sísmica nos instantes de tempo $0,24s$, $0,68s$ e $0,8s$, para uma fonte sísmica na posição $x = 750m$ e $z = 300m$. É possível notar a reflexão causada pelo contraste presente no modelo de velocidades. Nota-se também que a onda sísmica é fortemente atenuada nas bordas do modelo. Os efeitos de borda, decorrentes da modelagem acústica por diferenças finitas, devem ser evitados ao máximo para para que se mantenha a qualidade da modelagem e migração dos dados. É possível observar também que as dimensões da malha do campo de velocidades não são as mesmas dos snapshots. No Capítulo 2 (Metodologia) a

relação entre as dimensões da malha do campo de velocidade e as dimensões dos snapshots são melhor definidas, e as condições de borda melhor detalhadas.

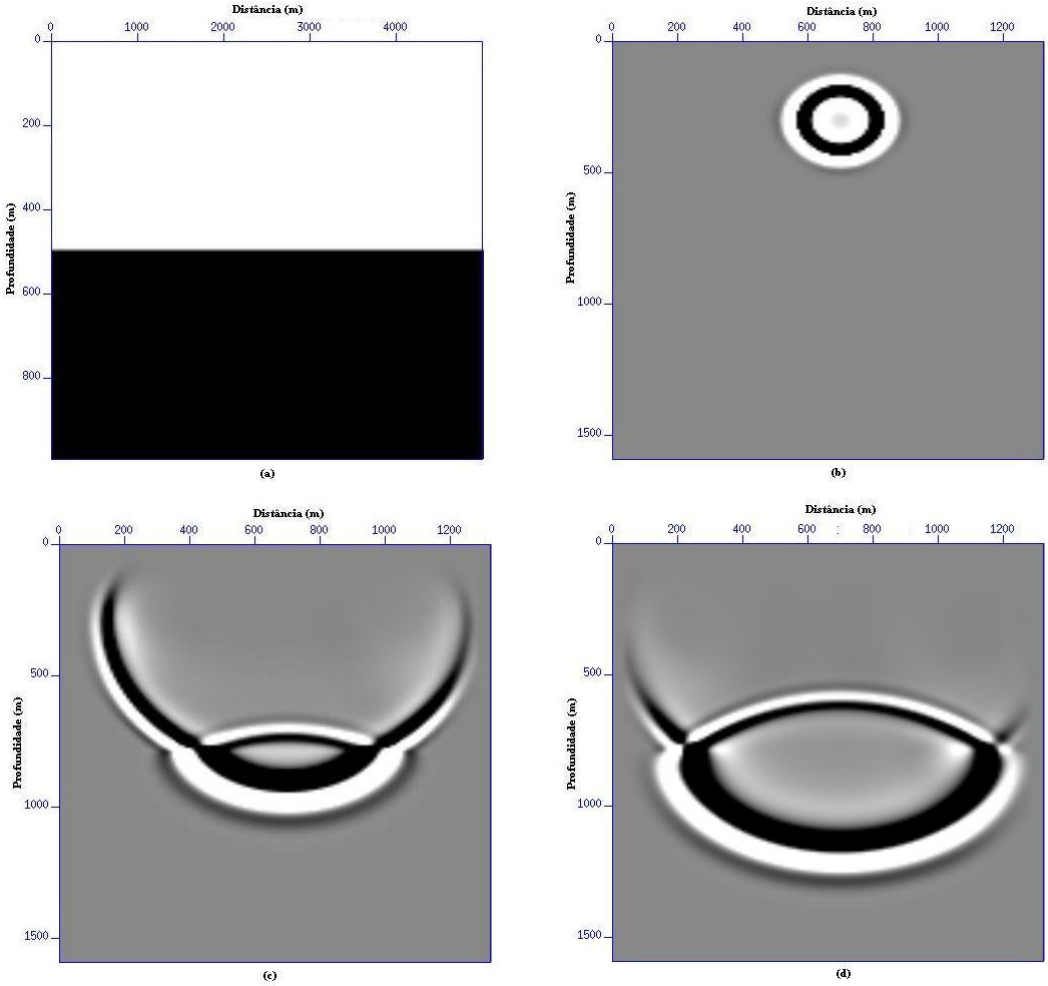


Figura 1.1: Modelagem sísmica. (a) Modelo de velocidades; (b) modelagem em 0,24s; (c) modelagem em 0,68s; (d) modelagem em 0,8s.

1.1.2 Diferenças Finitas

O método de diferenças finitas substitui as operações diferenciais da equação acústica completa da onda (equação 1.1) por aproximações utilizando truncamento da série de Taylor (dos Santos, 2010). Este método pode ser tanto utilizado para aproximar as derivadas temporais de $P(x, z, t)$ quanto seu laplaciano $\nabla^2 P$. A equação 1.3 é apenas uma formulação inicial para entendermos o problema da extrapolação temporal; neste trabalho, a solução da equação de onda será feita através do Rapid Expansion Method (REM, seção 1.4). Já o laplaciano $\nabla^2 P(x, z, t)$ é aproximado por um operador de diferenças finitas de quarta ordem.

Como citado, o operador de diferenças finitas é baseado no truncamento da série de Taylor que aproxima a derivada de segunda ordem, e quanto mais termos da série utilizados,

mais preciso é o resultado, porém mais cálculos são necessários. Por se tratar de um método numérico de aproximação, deve-se levar em conta condições de estabilidade e dispersão, de forma que os resultados gerados pela equação discretizada da onda (equação 1.3) sejam consistentes com a equação completa da onda acústica (equação 1.1). Devido a estas condições de estabilidade e dispersão, a aproximação por esquemas de alta ordem (neste caso, quarta ordem) requer que o passo de avanço no tempo seja reduzido (Lines, Slawinski e Bording, 1999). A seção 1.3 descreve de forma mais detalhada como os espaçamentos espaciais discretos $(\Delta x, \Delta z)$ dependem do espaçamento temporal Δt devido às condições de estabilidade e dispersão.

Por exemplo, a derivada segunda em x é aproximada pelo seguinte operador de diferenças finitas de segunda ordem:

$$\frac{\partial^2 P(x,z,t)}{\partial x^2} \approx \frac{P(x+\Delta x,z,t) - 2P(x,z,t) + P(x-\Delta x,z,t)}{\Delta x^2} \tag{1.4}$$

Neste trabalho, o operador utilizado é de quarta ordem, ou seja, a derivada em x é aproximada por:

$$\begin{aligned}\frac{\partial^2 P}{\partial x^2} \approx \frac{-1}{12(\Delta x^2)}[P(x+2\Delta x,z,t) - 16[P(x+\Delta x,z,t)+ \\ P(x-\Delta x,z,t)] + 30P(x,z,t) + P(x-2\Delta x,z,t)]\end{aligned} \tag{1.5}$$

similarmente, para a derivada em z,

$$\begin{aligned}\frac{\partial^2 P}{\partial z^2} \approx \frac{-1}{12(\Delta z^2)}[P(x,z+2\Delta z,t) - 16[P(x,z+\Delta z,t)+ \\ P(x,z-\Delta z,t)] + 30P(x,z,t) + P(x,z-2\Delta z,t)]\end{aligned} \tag{1.6}$$

e o laplaciano $\nabla^2 P$ é aproximado por

$$\nabla^2 P = \frac{\partial^2 P}{\partial x^2} + \frac{\partial^2 P}{\partial z^2} \approx D_4^2[P](x) + D_4^2[P](z) \tag{1.7}$$

onde $D_4^2[P](x)$ e $D_4^2[P](z)$ significam aproximação por diferenças finitas da derivada segunda com um operador de quarta ordem, correspondentes às equações 1.5 e 1.6, respectivamente. As deduções mais detalhadas para as equações de diferenças finitas de segunda e quarta ordem desta seção, baseadas no truncamento da série de Taylor, podem ser encontradas em Araújo (2009).

1.1.3 Condições de estabilidade e dispersão do Método de Diferenças Finitas

Utilizando-se o esquema de diferenças finitas onde a derivada segunda no tempo é aproximada pela equação 1.2 e as derivadas espaciais pelas equações 1.5 e 1.6, é preciso respeitar uma determinada condição de estabilidade, do contrário o método poderá gerar valores infinitamente grandes (dos Santos, 2010). De forma resumida, a instabilidade decorre da discretização da equação da onda, na qual a aproximação dos infinitesimais dx, dz e dt por espaçamentos discretos $\Delta x, \Delta z$ e Δt implica numa equação que relaciona e restringe estes parâmetros discretos, para evitar erro numérico.

A relação a ser obedecida para evitar instabilidade numérica é dada por Lines, Slawinski e Bording (1999):

$$\Delta t_{max} < \frac{2\sqrt{\frac{2}{1/\Delta x^2 + 1/\Delta z^2}}}{c_{max}\sqrt{\mu}} \tag{1.8}$$

onde μ é a soma dos valores absolutos dos coeficientes do operador de diferenças finitas, ou seja:

$$\mu = \sum_{-n/2}^{n/2} |w_x| + |w_z| \tag{1.9}$$

Para o método de quarta ordem (equações 1.5 e 1.6), n=4 e w_x, w_z são operadores de cinco pontos dados por $1/12(-1, 16, -30, 16, -1)$, resultando em $\sqrt{\mu} = \sqrt{32/3}$. A variável c_{max} representa a maior velocidade sísmica do modelo no qual se quer realizar a propagação da onda. Assim, a equação 1.8 mostra que, quanto maior a precisão das aproximações das derivadas espaciais, maior será o operador w, e maior será a soma μ dos seus coeficientes, o que obriga à redução no passo temporal máximo Δt_{max}. É também possível notar que, quanto menores os espaçamentos $\Delta x, \Delta z$ tanto menor deve ser Δt_{max}. Sabe-se que quanto menores os espaçamentos espaciais, e quanto maior for a ordem da aproximação do laplaciano, mais precisos os resultados; o custo disto, porém, é a redução do passo temporal. Na prática isso significa um aumento das iterações no tempo conforme o campo de onda se propaga, ou seja, aumento do tempo computacional tanto na modelagem quanto na RTM. O dilema "precisão vs. custo computacional" é mais um estímulo ao uso de algoritmos paralelizados, como aqueles que fazem uso de placas gráficas (GPU), explorados neste trabalho (ver Capítulo 3).

Já o problema de dispersão numérica ocorre quando a velocidade de fase da onda, isto é, a velocidade de propagação de cada componente de frequência, é diferente da velocidade de grupo. Apesar de não afetar a estabilidade da propagação, a dispersão numérica gera resultados ruidosos, que prejudicam a comparação do dado modelado com o dado observado. No caso da equação da onda, a dispersão é causada pela dificuldade do método de diferenças

finitas em representar corretamente a forma da onda se propagando no espaço discretizado (dos Santos, 2013). Para evitar esse problema, é preciso respeitar uma relação, conhecida como condição de dispersão, que normalmente é dada por:

$$f_{max} \leq \frac{min[c(x,z)]}{Gmax(\Delta x, \Delta z)} \tag{1.10}$$

com f_{max} sendo a frequência máxima da onda propagada e G sendo um número constante que dependerá da ordem do operador de diferenças finitas usado no cálculo das derivadas espaciais. Quanto menos exato for o operador, mais pontos por comprimento de onda deverão ser amostrados e, consequentemente, maior o G. Para um operador de segunda ordem, por exemplo, G assume valor 10, já um operador de oitava ordem pode ter $G = 4$ ou até mesmo 3. O menor valor assumido por G será 2, para operadores mais longos (dos Santos, 2013).

1.1.4 Método de Expansão Rápida (REM)

Para permitir uma melhor aproximação da derivada segunda no tempo, além de prover uma base para uma solução recursiva que permite o avanço no tempo em grandes intervalos, utilizamos o método de expansão rápida, ou *Rapid Expansion Method* (Pestana e Stoffa, 2010).

Na implementação do REM utilizada aqui, as derivadas espaciais são calculadas por aproximações de diferenças finitas de quarta ordem, como descrito na seção 1.1.1. Este método usa a solução exata da equação da onda, dada por:

$$P(x,z,t+\Delta t) = 2cos(L\Delta t)P(x,z,t) - P(x,z,t-\Delta t) + S(x,z,t) \tag{1.11}$$

onde $L^2 = -c^2(x,z)\nabla^2$. Pode-se notar a grande semelhança da equação (1.11) com a equação para aproximação da derivada temporal (1.2) apresentada na seção 1.1.

A solução da equação (1.11) se reduz à expansão da função cosseno. Segundo a abordagem apresentada por Tal-Ezer, Kosloff e Koren (1987) a função cosseno pode ser expandida da seguinte forma:

$$cos(L\Delta t) = \sum_{k=0}^{M} C_{2k} J_{2k}(\Delta t R) Q_{2k}(\frac{iL}{R}) \tag{1.12}$$

onde $C_{2k} = 1$ para $k = 0$ e $C_{2k} = 2$ para $k > 0$. O valor J_{2k} representa a função de Bessel de ordem $2k$, e $Q_{2k}(w)$ são os polinômios modificados de Chebyshev, calculados segundo a relação recursiva:

$$Q_{k+2}(w) = 2(1 + 2w^2)Q_k(w) - Q_{k-2}(w) \tag{1.13}$$

que é iniciada por $Q_0(w) = 1$ e $Q_2(w) = 1 + 2w^2$, onde $w = iL/R$ (dos Santos, 2010).

O termo R é um escalar associado ao maior autovalor de $-L^2$. Para o caso 2-D, o valor de R é dado por $R = \pi c\sqrt{(1/\Delta x^2) + (1/\Delta z^2)}$. Em geral, o valor de c deve ser substituído por c_{max}, o maior valor de velocidade do modelo (Pestana e Stoffa, 2010).

A estabilidade do REM é garantida pelo somatório 1.12, computando o número máximo de termos adequado ao problema. Além disso, a série converge exponencialmente para $M > R\Delta t$, isto é, o número de termos usados na expansão é diretamente proporcional ao intervalo de amostragem no tempo utilizado. Isso significa que qualquer que seja o Δt, o número de termos calculados será suficiente para garantir a convergência da expansão, e, consequentemente, a estabilidade numérica do método (dos Santos, 2013).

No trabalho de Araújo (2009), é provada a equivalência entre as expressões do REM e do método de diferenças finitas: fazendo $M = 2$ em 1.12, é obtida a expressão para a equação da onda aproximada pelo operador de diferenças finitas de segunda ordem; fazendo $M = 4$, se obtém a expressão para a equação da onda aproximada pelo operador de diferenças finitas de quarta ordem no tempo.

1.2 Migração Reversa no Tempo Pré-empilhamento

No processamento sísmico cujo objetivo é obter uma imagem final que se assemelhe à geologia da subsuperfície, a etapa de migração é imprescindível em meios de geologia complexa (refletores muito inclinados, dobras, falhas, entre outras feições). Como descrito na Introdução, a migração sísmica move reflexões inclinadas para suas verdadeiras posições na subsuperfície e colapsa difrações, aumentando assim a resolução espacial e produzindo uma imagem sísmica mais fidedigna da supsuperfície. O objetivo da migração pós-empilhamento é fazer com que a seção empilhada se assemelhe à seção geológica em profundidade (Yilmaz, 2001). Todavia quando a subsuperfície apresenta elevada complexidade, como grandes contrastes laterais de velocidade, falhas, dobras e diápiros, estas feições não são bem imageadas, e é necessário que se realize a migração pré-empilhamento em profundidade - no caso deste trabalho, a migração reversa no tempo pré-empilhamento.

Algoritmos de migração pré-empilhamento em profundidade são divididos em duas amplas categorias: aqueles baseados na teoria do raio e os outros métodos com base na equação da onda. Cada grupo é novamente dividido em subgrupos; os métodos baseados na teoria do raio incluem migração Kirchhoff e Migração de Feixes Gaussianos (*Beam migration*).

Aqueles baseados na equação da onda subdividem-se nos que usam a equação da onda unidirecional e os que utilizam a equação da onda acústica completa; a migração reversa no tempo (RTM, *Reverse Time Migration*) está incluída neste último subgrupo. Os métodos de migração baseados na teoria do raio não apresentam bons resultados em modelos complexos de velocidade. Já os que utilizam a equação da onda unidirecional não conseguem lidar com a propagação da onda em grandes ângulos (iguais ou maiores que 90^o), sendo ineficazes no imageamento de refletores de alto mergulho. Em contrapartida, a migração reversa no tempo utiliza a equação completa da onda acústica para realizar a propagação do campo de onda, e é capaz de imagear meios geológicos de alta complexidade; no imageamento de bacias sedimentares de geologia complexa, como no Golfo do México, a RTM é o algoritmo de preferência dada sua superioridade em relação aos métodos previamente citados (Liu et al., 2011).

A migração reversa no tempo foi inicialmente investigada na década de 1980 (Baysal, Kosloff e Sherwood, 1983a; Loewenthal e Mufti, 1983; Whitmore, 1983; McMechan, 1983). Entretanto, durante algum tempo esta investigação foi apenas teórica ou testada somente para modelos simples, devido ao seu elevado custo computacional, que inviabiliziou uma aplicação mais extensa do método na época. Atualmente os processadores estão muito mais rápidos, mas ainda assim, em um único processador, a migração reversa no tempo de um dado 2D, como por exemplo o Marmousi (disponibilizado pelo IFP), de dimensões espaciais 369×375, 725 amostras temporais por traço, 240 tiros e 96 canais por tiro leva um tempo de aproximadamente $7h15min$, com base nos resultados obtidos por dos Santos (2010). De forma a realizar o processamento em menor tempo, neste trabalho foram implementados algoritmos que fazem uso de computação paralela, em especial unidades de processamento gráfico (GPUs). A RTM implementada desta maneira, para o mesmo modelo e utilizando parâmetros similares, levou um tempo aproximado de $5min54s$, permitindo uma maior liberdade de testes e geração de resultados em tempo hábil.

1.2.1 Modelo dos refletores explosivos

Os métodos convencionais de migração de dados empilhados se baseiam na premissa que a seção sísmica em tempo se adequa ao *modelo dos refletores explosivos* (Claerbout, 1976). Este modelo parte do pressuposto que toda energia presente num tempo t na seção empilhada se origina por reflexões no tempo $\frac{1}{2}t$, e assim o valor do coeficiente de reflexão pode ser determinado se propagarmos esta energia de volta à subsuperfície até o marco $\frac{1}{2}t$, ou, equivalentemente, propagar esta energia do tempo t ao tempo 0 utilizando a metade das velocidades de propagação da subsuperfície (Levin, 1984).

Uma argumentação mais analítica acerca do uso do modelo dos refletores explosivos pode ser obtida se observarmos a solução da equação acústica da onda (1.1). Se pode

observar que a propagação dos dados no tempo pode ser executada como um avanço ou como um retrocesso (depropagação) no tempo. Isto é, se $P(x, z, t)$ é uma solução da equação (1.1), então necessariamente $P(x, z, t_0 - t)$ é também solução, para qualquer t_0 constante (Levin, 1984). Este argumento é uma das bases para a geração final da imagem sísmica.

1.2.2 Algoritmo da migração reversa no tempo pré-empilhamento

O algoritmo da migração reversa no tempo pré-empilhamento se divide em três etapas:

1. Extrapolação direta no tempo (t_0 a t_{max}) do campo de onda da fonte $P_s(x, z, t)$;
2. Extrapolação reversa no tempo (t_{max} a t_0) do campo de onda do receptor $P_r(x, z, t)$;
3. Aplicação de uma condição de imagem adequada à construção da imagem onde ocorreram reflexões.

Estas três etapas são realizadas para cada tiro. A etapa (1) corresponde à modelagem (na forma descrita na seção anterior) da fonte na posição correspondente à família de tiro comum que se está migrando. A etapa (2) corresponde à extrapolação reversa no tempo das amplitudes dos traços desta mesma família de tiro. Em ambas as etapas a extrapolação é realizada com base na equação da onda acústica (equação 1.1), a qual neste trabalho tem suas derivadas espaciais aproximadas por operadores de diferenças finitas de quarta ordem (equação 1.7) e a extrapolação no tempo através do método de expansão rápida (equação 1.11). O passo temporal é realizado com a equação 1.11 tanto na extrapolação direta quanto na reversa (neste caso, é calculado o termo $P(x, z, t - \Delta t)$).

A imagem é convencionalmente construída tomando a correlação cruzada dos campos de onda extrapolados da fonte $P_s(x, z, t)$ e do receptor $P_r(x, z, t)$ (Liu et al., 2007):

$$I(x, z) = \int_0^{t_{max}} P_s(x, z, t) P_r(x, z, t) dt, \tag{1.14}$$

que na forma discreta, implementada no algoritmo, é dada por

$$I(x, z) = \sum_{t=0}^{t_{max}} P_s(x, z, t) P_r(x, z, t). \tag{1.15}$$

A equação 1.14 fornece a cinemática correta de uma reflexão na qual os campos de onda incidente e refletido coincidem no espaço e no tempo (de acordo com a idéia de refletor explosivo mostrada na subseção 1.2.1). A Figura 1.2, retirada de Liu et al. (2011), mostra raios (perpendiculares às frentes de onda extrapoladas) de uma fonte e um receptor; no ponto de encontro "I" a correlação é máxima e o refletor é imageado.

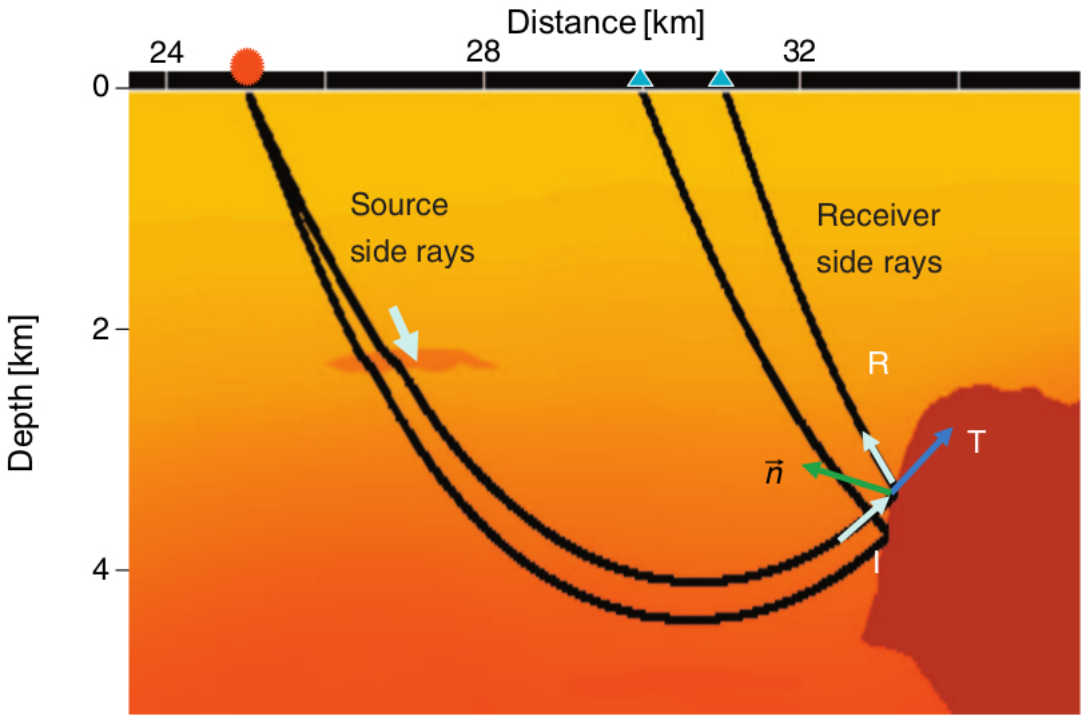


Figura 1.2: Exemplo de alguns raios para uma fonte e dois receptores. A imagem é construída no ponto "I" indicado (retirado de Liu et. al., 2011).

Voltando à condição de imagem (equação 1.14), pode-se observar que o campo $P_s(x, z, t)$ e o campo $P_r(x, z, t)$ são extrapolados em direções opostas no eixo do tempo, isto é, o campo de onda da fonte é extrapolado à frente no tempo, começando no tempo zero e na posição da fonte, enquanto que o campo de onda do receptor é extrapolado de forma reversa no tempo, começando no tempo máximo de registro dos receptores. A inserção dos campos na equação 1.14 para o posterior cálculo da correlação podem ser abordados de três maneiras: salvar todos os snapshots do campo de onda $P_s(x, z, t)$ e durante o cálculo reverso no tempo de $P_r(x, z, t)$, acessar na memória as posições previamente salvas de $P_s(x, z, t)$; ou salvar snapshots em intervalos de tempo discretos maiores que o passo temporal, recuperando os snapshots restantes de $P_s(x, z, t)$ durante a condição da imagem, através de interpolação ou extrapolação (técnica de *checkpointing*, Symes, 2007); ou uma terceira opção, que é salvar os valores do campo de onda apenas nas bordas da malha para todos os passos de tempo, e depois computar o campo de onda nos pontos da parte interna da malha utilizando os valores salvos das bordas como condições de contorno. Cada uma destas formas tem suas vantagens e desvantagens, a depender da configuração do sistema (Dussaud et al., 2008). Como a GPU possui memória suficiente, no caso dos modelos 2D testados neste trabalho, para salvar toda a matriz $P_s(x, z, t)$ e realizar a condição de imagem, optou-se pela primeira opção, que apesar de requerer mais memória é a mais precisa.

1.2.3 Ruído de baixa frequência e filtro laplaciano

Existe um ruído presente em toda seção migrada por RTM, que é decorrente da condição de imagem. Este ruído é gerado devido à correlação das componentes de baixa frequência dos campos de onda (fonte-receptor) em pontos que não correspondem a refletores sísmicos. Como resultado disso a seção migrada perde resolução, como mostra a Figura 1.3.

De forma a contornar esse problema, neste trabalho foi aplicado um filtro laplaciano na imagem final. Esse filtro consiste em susbstituir a amplitude da imagem em cada ponto pelo seu laplaciano:

$$P_{corr}(x,z,t) = \frac{\partial^2 P(x,z,t)}{\partial x^2} + \frac{\partial^2 P(x,z,t)}{\partial z^2}. \tag{1.16}$$

O resultado da aplicação do filtro laplaciano é mostrado na Figura 1.4. Na implementação deste trabalho, as derivadas do filtro laplaciano são aproximadas por operadores de diferenças finitas de segunda ordem. Todas as imagens de seções migradas aqui apresentadas (exceto a Figura 1.4) foram filtradas dessa maneira.

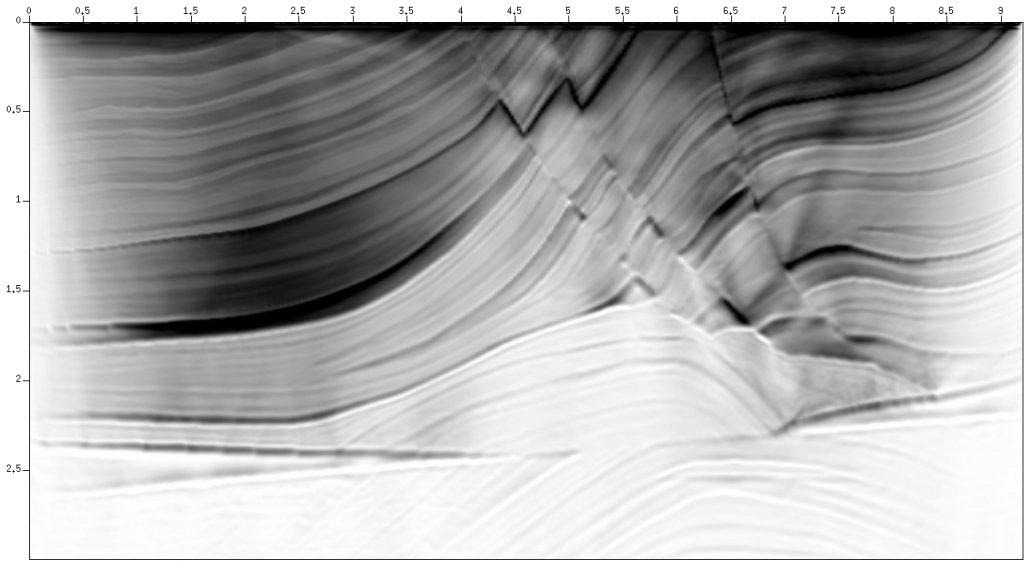


Figura 1.3: Seção migrada sem filtro laplaciano, modelo Marmousi.

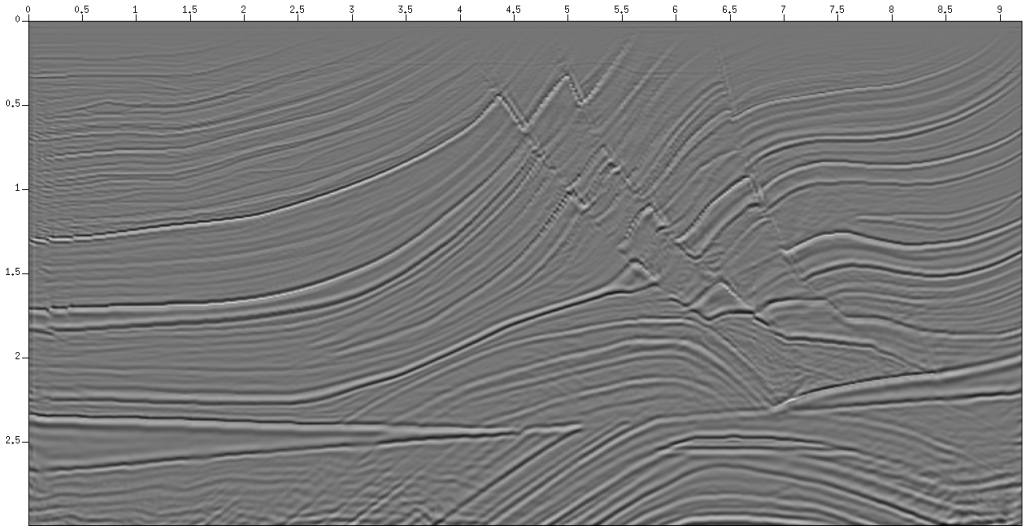


Figura 1.4: Mesma seção da figura anterior, após filtro laplaciano.

CAPÍTULO 2

Metodologia

Neste capítulo são detalhados os passos fundamentais dos algoritmos de modelagem sísmica e RTM. Como a RTM consiste, do ponto de vista prático, de duas modelagens (uma propagação e uma depropagação), mais a condição de imagem, a análise da metodologia da modelagem, na seção seguinte, é frutífera já que esta é uma das bases para realização da migração reversa no tempo.

2.1 Metodologia da modelagem sísmica

A modelagem sísmica de tiros consiste nas seguintes etapas:

1. Aloca uma malha 2-D com profundidade z_{max} igual à do campo de velocidades, e extensão x_{max} igual ao tamanho do arranjo que se quer modelar. Faz $P(x, z, t) = 0$ em toda a malha inicialmente;

2. Expande x_{max} nas duas direções laterais (respeitando os limites do modelo de velocidades), pois podem haver reflexões decorrentes de feições que não estão diretamente abaixo do arranjo;

3. Expande o novo x_{max} e o z_{max} em todas as direções, que será a borda do modelo. A borda tem a importante função de atenuar reflexões geradas nos limites do modelo, intrínsecas à implementação numérica da propagação da onda numa malha finita;

4. Insere o pulso (*wavelet* Ricker) na posição (x_s, z_s) determinada;

5. Aplica a equação da onda 1.11 em cada ponto da malha, iterando no tempo;

6. (Opcional) Grava todos os valores de $P(x, z, t)$, ou seja, snapshots, a cada 100 (exemplo) iterações no tempo;

7. A cada iteração temporal, grava $P(x, z = 0, t_{atual})$ num vetor, compondo a seção sísmica sequencialmente.

8. Ao fim das iterações temporais, move o arranjo e modela o tiro seguinte, voltando à etapa 1.

Na Figura 2.1 temos o modelo de velocidades sintético 2-D do dado Marmousi, com $9,225km$ de distância e $3km$ de profundidade, com intervalos de amostragem $25m$ e $8m$, respectivamente. A malha da etapa **(1)** é mostrada na Figura 2.2. Percebe-se que a malha possui a mesma dimensão vertical do modelo de velocidades, mas a dimensão horizontal tem o tamanho do arranjo de geofones desejado.

Na etapa **(2)** expande-se a malha lateralmente, de forma a captar reflexões decorrentes de feições laterais que não estejam diretamente abaixo do arranjo de geofones, como mostra a Figura 2.2.

Na etapa **(3)** é aplicada a condição de borda. Como o algoritmo de diferenças finitas interpreta o final da malha como um forte contraste de velocidades, gerando reflexões nesses limites, cria-se uma borda (em cinza na Figura 2.2) ao redor da malha. Nesta borda, é aplicado um decaimento exponencial gradual, chamado de *taper*, no campo P (mais intenso nas partes mais internas da borda, menos intenso conforme se aproxima dos limites da borda). Desta forma, quando as ondas chegam nos limites mais externos da borda, suas amplitudes estão tão pequenas que as reflexões geradas nos limites podem ser desprezadas.

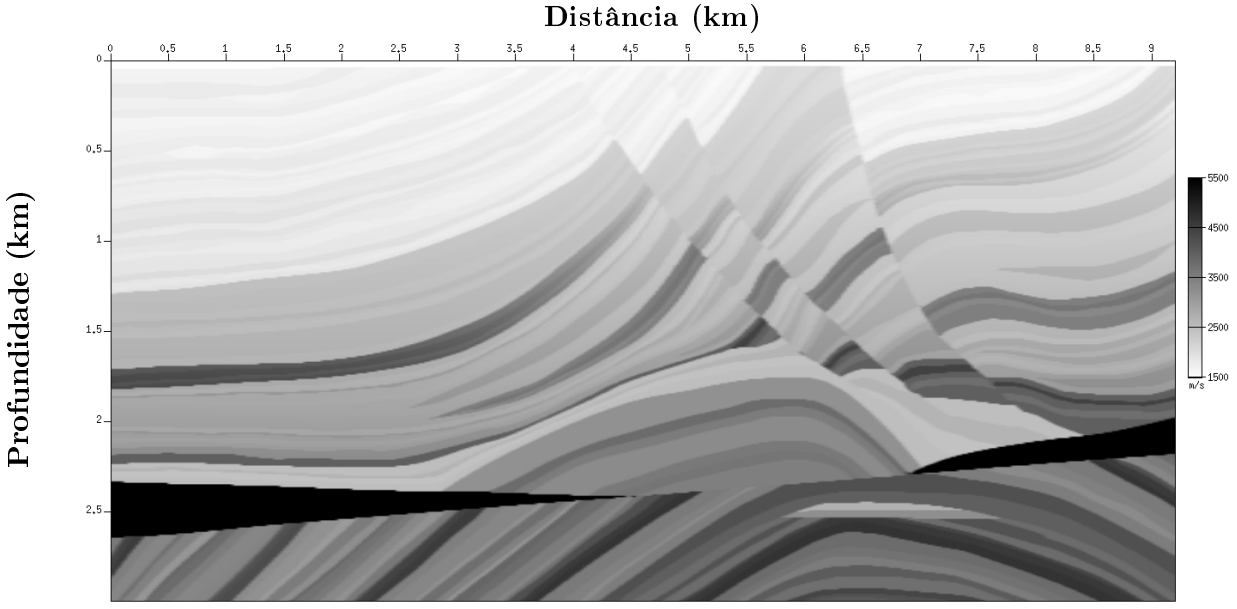


Figura 2.1: Modelo de velocidades do dado Marmousi

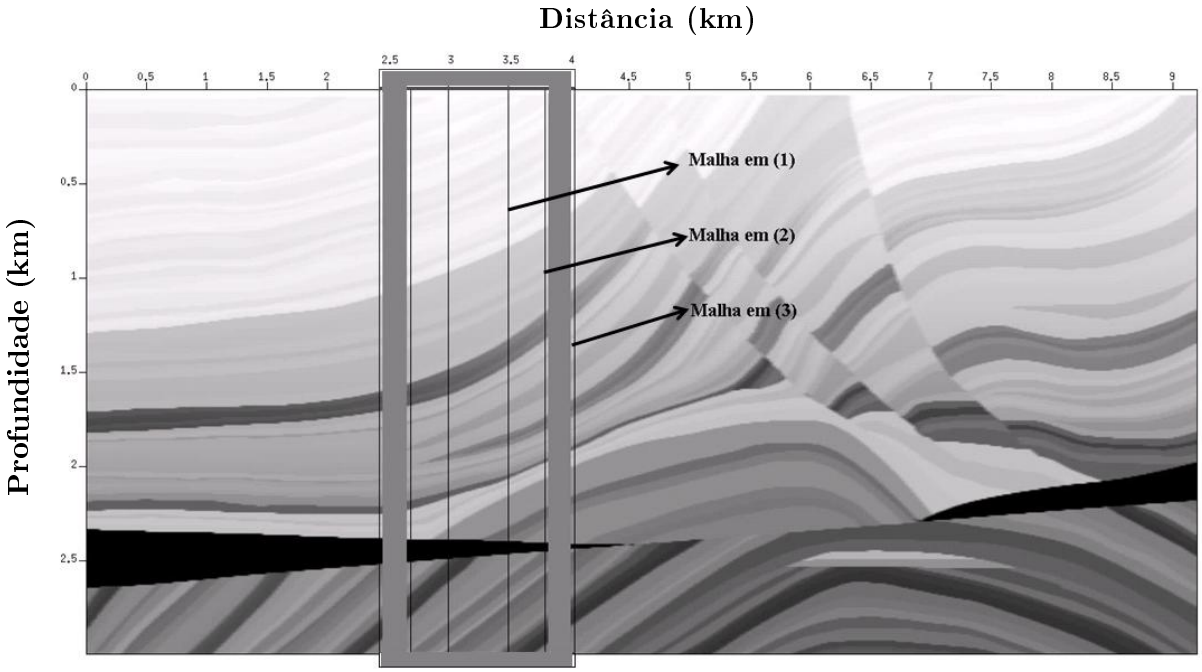


Figura 2.2: Malhas do campo de onda P, em relação ao modelo de velocidades, nas etapas (1), (2) e (3) (fora de escala).

A Figura 2.3 é similar à Figura 1.1 do Capítulo 1, porém sem aplicação do taper nas bordas. Como se pode perceber, ocorrem várias reflexões nas bordas conforme se avança no tempo, que são registradas nos geofones; a migração com estas reflexões de borda gera uma imagem de qualidade ruim, daí a importância de aplicar a condição de borda absorvedora.

A etapa **(4)** marca o início do processo de modelagem, através da inserção da fonte no modelo. A etapa **(5)** é aplicada em cada iteração temporal, propagando o pulso pelo modelo.

A etapa **(6)** é opcional e bastante custosa, pois envolve gravar a matriz de pressões $P(x, z, t)$ desde t_0 até t_{final}, a cada certo número de iterações temporais (pois gravar a cada iteração seria ainda mais custoso). Esta opção é vantajosa quando se quer avaliar a propagação da onda no modelo. Verificar se o taper está funcionando, por exemplo, pode ser feito através da análise de snapshots da Figura 1.1(com taper) e da Figura 2.3(sem *taper*). Porém, por ser muito custoso gravar matrizes a cada iteração, esta etapa é omitida na geração de sismogramas (tiros).

A etapa **(7)** é a gravação do sismograma, onde $P(x, z = 0, t = t_{atual})$ é gravado a cada iteração temporal. O resultado pode ser visto na Figura 2.4. Como esta modelagem foi feita com taper, as únicas feições que aparecem são a onda direta (próxima à superfície) e a onda refletida (marca de $1s$).

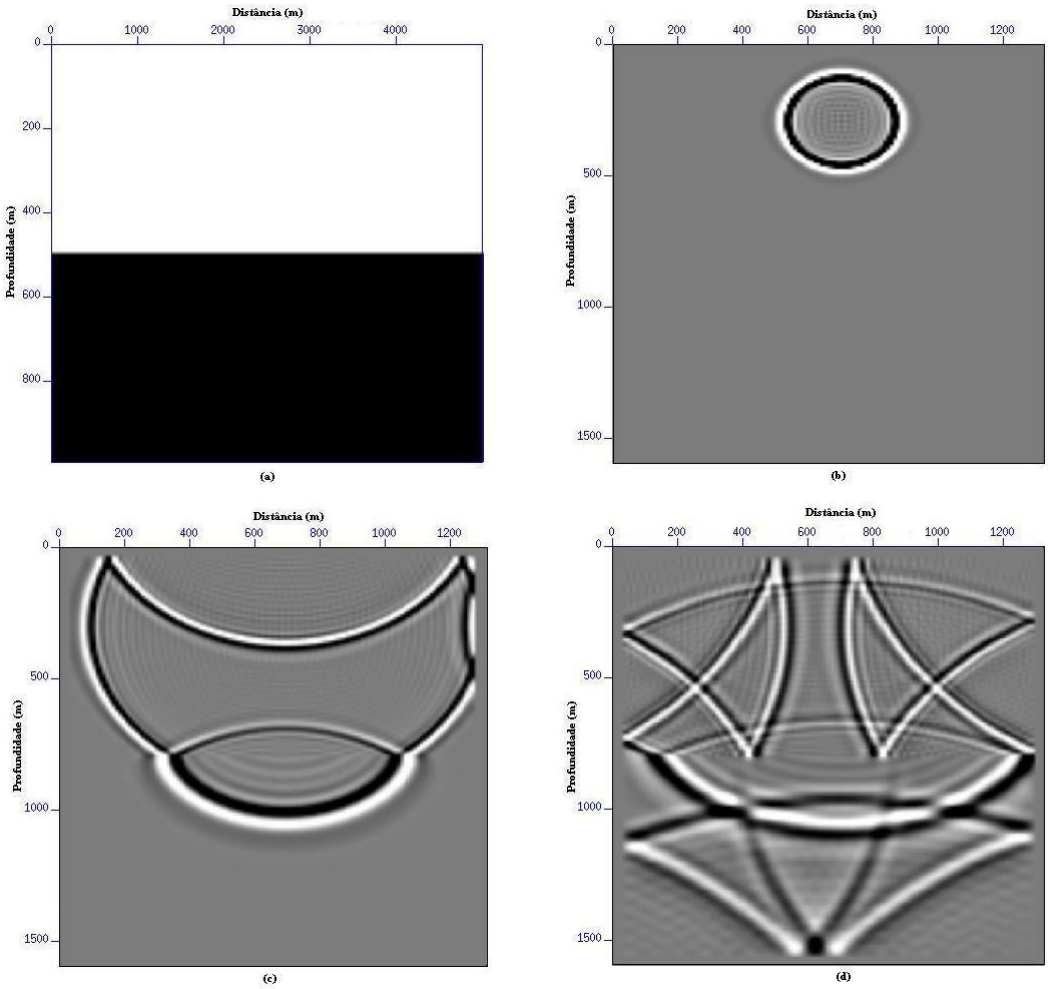


Figura 2.3: Modelagem sísmica sem *taper*. (a) Modelo de velocidades, (b) modelagem em 0,24s, (c) modelagem em 0,68s, (d) modelagem em 1,24s.

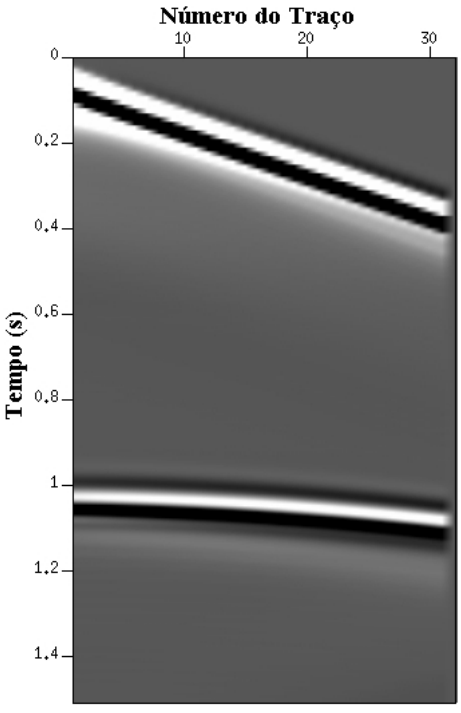


Figura 2.4: Sismograma resultante da modelagem (com *taper*) no modelo de 2 camadas, para arranjo end-on com 32 geofones.

Por fim, a etapa **(8)** consiste na repetição de todos os passos anteriores, mas movendo o arranjo de geofones, de forma a modelar o próximo tiro. O processo termina ao se chegar no tiro final.

2.2 Metodologia da migração reversa no tempo

A migração reversa no tempo implementada neste trabalho segue os seguintes passos:

1. Acessa o header da família de tiro que se está migrando e obtém as posições dos geofones e fonte para o tiro atual. Copia o campo de velocidades para o programa e aloca a malha na qual será propagada a onda, similar à malha **3** da Figura 2.2, com base na geometria do tiro atual;

2. Faz uma modelagem sintética com os mesmos parâmetros (posições de geofone e fonte) do tiro atual. Começa as iterações no tempo (de $t = 0$ a $t = t_{max}$):

   - Insere a fonte em sua posição nos tempos iniciais;
   - Propaga o campo de onda pelo modelo, aplicando a equação do REM em cada ponto da malha;
   - Grava numa matriz 3D o estado atual do campo de pressões, $MOD(x, z, t_{atual})$. Ao final a matriz conterá todos os snapshots desde $t = 0$ a $t = t_{max}$, ou seja, $MOD(x, z, t)$;

3. Inicia a migração reversa no tempo. Começa um novo loop de iterações temporais, partindo de t_{max} até $t = 0$:

   - Aloca uma nova malha, com as mesmas dimensões. Insere os valores de $P(x, z, t = t_{max})$ provenientes do tiro atual;
   - Aplica o REM depropagando no tempo, salva o snapshot em $SHOT(x, z, t_{max} - \Delta t)$;
   - Aplica a condição de imagem:
     $I_j(x, z) = I_{j-1}(x, z) + SHOT(x, z, t_{max} - j\Delta t) * MOD(x, z, t_{max} - j\Delta t)$.
     A imagem $I_j(x, z)$ é atualizada a cada iteração j, de forma que ao chegar em $t = 0$ é obtida a imagem completa para o tiro atual.

4. Volta à etapa 1 com o próximo tiro, e repete o processo. Finaliza ao chegar no último tiro.

A etapa **(1)** é exemplificada pela Figura 2.5. Nesta figura vemos o primeiro tiro do dado Marmousi; informações do header mostram que o primeiro geofone do primeiro tiro

se encontra na posição $x = 0,425km$, e a sua fonte em $x = 3km$. Com estas informações é delimitada a primeira malha, mostrado na Figura 2.5. Esta malha ainda será expandida lateralmente duas vezes, e será semelhante à malha **3** da Figura 2.2.

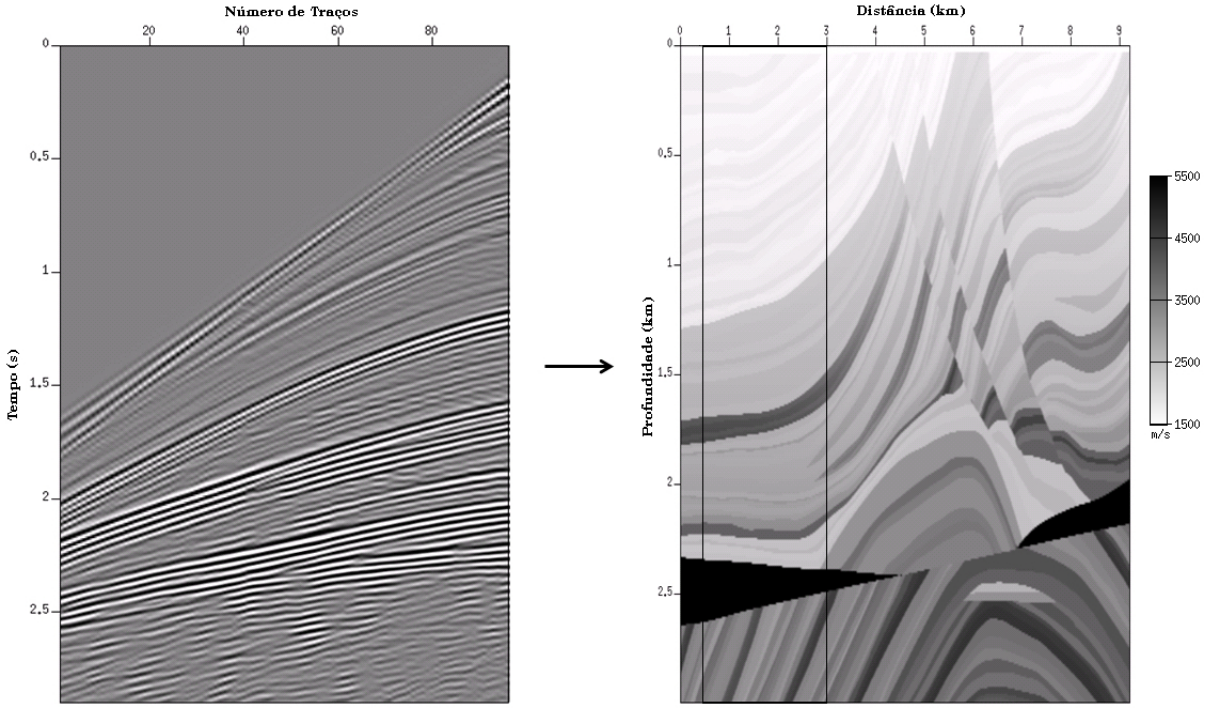


Figura 2.5: Seleção da fatia do campo de velocidades para a malha de modelagem, a partir dos dados de tiro (modelo Marmousi)

Na etapa **(2)** o campo de onda é propagado, calculando $P(x, z, t - \Delta t)$ em cada ponto da malha. Para efetuar o cálculo de $P(x, z, t - \Delta t)$ é utilizada a equação 1.11, ou seja, em cada ponto da malha é preciso calcular o laplaciano (por diferenças finitas de quarta ordem), o termo cosseno (por expansão de polinômios de Chebyshev), multiplicado pelo campo em $P(x, z, t)$, e por fim somar o termo $P(x, z, t + \Delta t)$. Este conjunto de operações está associado a cada ponto da malha, a cada iteração temporal, e é a etapa que tem maior custo de processamento na RTM. Enquanto que numa implementação em série teriam de ser feitos estes conjuntos de operações para o cálculo de $P(x, z, t - \Delta t)$ $nx \times nz$ vezes, sendo nx e nz as dimensões da malha, na implementação em GPU é possível designar $nx \times nz$ unidades de processamento, onde cada uma é responsável por um ponto da malha, estimando todos os valores de $P(x, z, t - \Delta t)$ simultaneamente (no Capítulo 3 estas unidades -threads- e o processamento são explicados com maior detalhe). Posteriormente são salvos todos os snapshots da modelagem na matriz 3D $MOD(x, z, t)$, a serem usados futuramente na condição de imagem; este é um dos passos que mais requer memória.

Na etapa **(3)** se inicia a RTM. É alocada uma nova malha, idêntica à da etapa **(1)**, onde é inserido o campo de pressões do tiro, inicialmente (t_{max}). Não é preciso alocar uma matriz 3D como na modelagem, pois a depropagação é realizada em conjunto com a condição de

imagem, a cada iteração reversa no tempo, economizando memória. Ao final, com a imagem completa correspondente ao tiro atual, a matriz 3D com os snapshots da modelagem e a matriz 2D da depropagação são desalocadas, liberando memória para o imageamento do tiro seguinte.

Na etapa **(4)**, todo o processo é repetido, mas para o próximo tiro, movendo o arranjo para a direita; daí conclui-se que a imagem é construída tiro a tiro. As Figuras 2.6, 2.7 e 2.8 ilustram este processo (10 tiros, 100 tiros e 240 tiros); como se pode perceber, as fatias da imagem vão sendo adicionadas tiro a tiro, até chegar na imagem final, com 240 tiros.

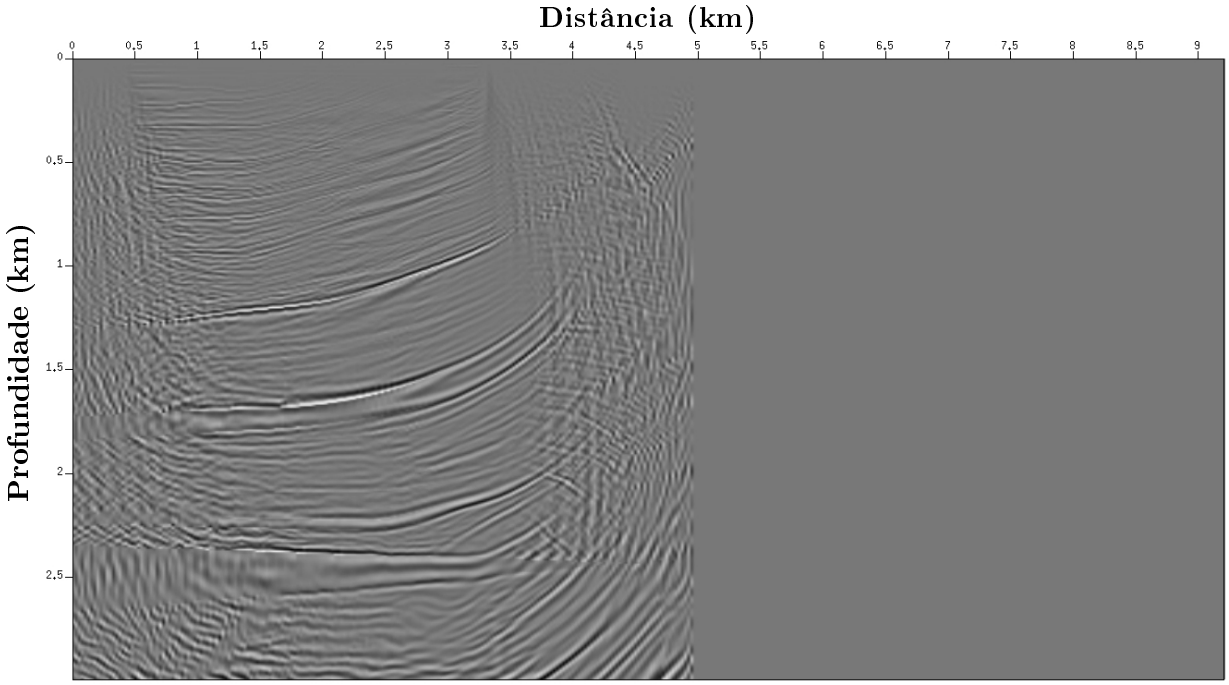


Figura 2.6: Imagem resultante da migração de 10 tiros para o modelo Marmousi.

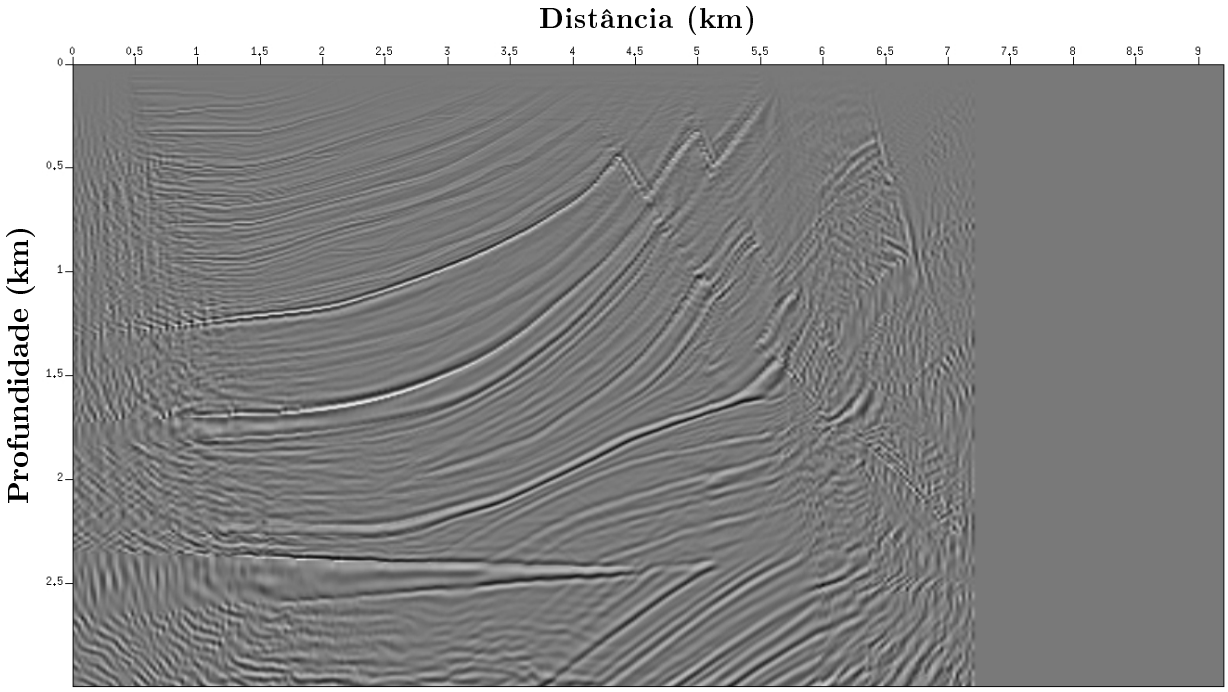


Figura 2.7: Imagem resultante da migração de 100 tiros para o modelo Marmousi.

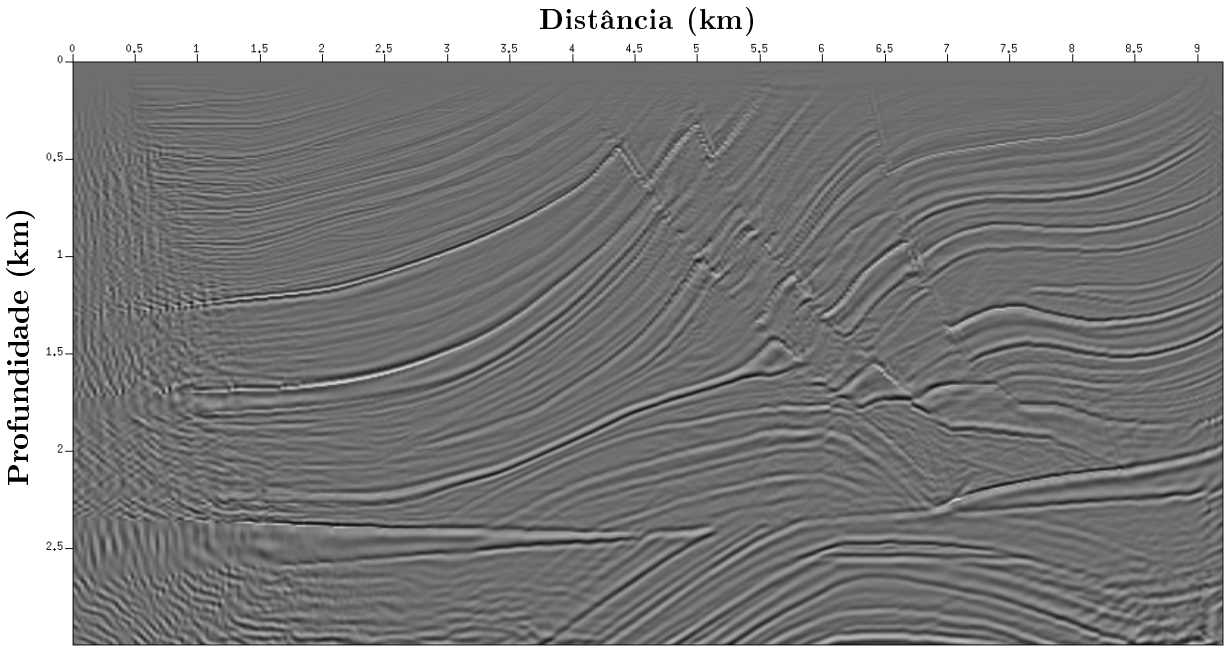


Figura 2.8: Imagem resultante da migração de 240 tiros para o modelo Marmousi.

CAPÍTULO 3

Programação paralela em unidades de processamento gráfico (GPUs)

Como mostrado no Capítulo 1, a migração reversa no tempo é baseada em algoritmos de modelagem sísmica, onde as derivadas são calculadas por esquemas de diferenças finitas ou pelo método de Fourier. A propagação (e depropagação) do campo de onda no tempo é geralmente feita usando-se diferenças finitas de baixa ordem. Quando as derivadas espaciais são calculadas por esquemas de alta ordem, existe um desbalanceamento que, normalmente, requer que o passo de avanço no tempo seja muito pequeno para garantir estabilidade e evitar dispersão numérica. Como resultado disso, pode-se usar intevalos maiores de amostragem no tempo (se necessário), e através do REM a instabilidade numérica é evitada quando a aproximação da série de Chebyshev é aplicada usando-se o número de termos apropriado. (Pestana and Stoffa, 2010).

Quanto ao problema computacional, existem duas formas de reduzir o tempo de execução de um programa serial sem realizar uma refatoração do código. A primeira é aumentar o número de operações por segundo usando um processador mais rápido. Já a segunda solução consiste em computar as partes seriais do código simultaneamente em paralelo. Assim, N processadores podem ser usados ao mesmo tempo reduzindo o tempo de processamento, sem exigir processadores mais rápidos.

Uma alternativa mais recente, implementada neste trabalho, é o uso de unidades de placas gráficas, ou *graphics processing unit* (GPU). As GPUs são originalmente unidades de processamento gráfico, por isso tem sua estrutura otimizada para a realização de operações de ponto flututante, como será detalhado neste capítulo.

Desta forma, a combinação do REM, que é o método considerado mais preciso para integração temporal, com uma implementação em GPUs, que é uma das tecnologias mais eficientes disponíveis, permite a execução de algoritmos para RTM e modelagem sísmica de uma forma mais precisa e também a um menor custo computacional.

3.1 A unidade de processamento gráfico - GPU

Devido à crescente demanda do mercado por imagens de cada vez mais alta-definição, associada à necessidade de transmissão em tempo real, a unidade de processamento gráfico (*Graphics Processing Unit*, ou GPU) evoluiu para uma estrutura altamente paralelizada, multifilamentada (por filamento ou *thread* entende-se sub-unidade de processamento) e com diversos núcleos, resultando numa tremenda capacidade de processamento de ponto-flutuante. A Figura 3.1 (NVIDIA-CUDA^{TM}, 2010) ilustra a superioridade das GPUs em velocidade de processamento (GFLOP/s):

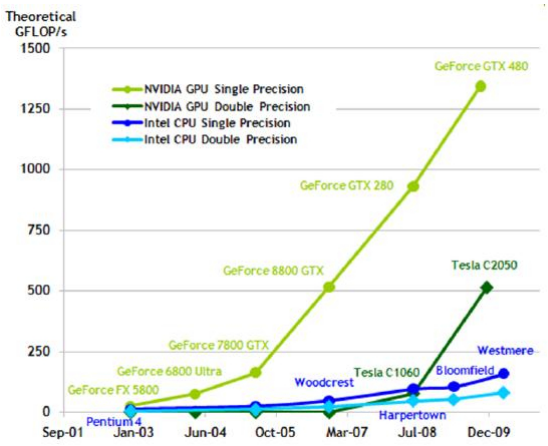


Figura 3.1: Operações em ponto flutuante (GFlops) - CPU e GPU ao longo dos últimos anos, retirado de $NVIDIA - CUDA^{TM}, 2010$

A razão por trás dessa discrepância entre CPU e GPU, no cálculo de pontos-flutuantes, é que a GPU é justamente especializada em computação intensiva e altamente paralelizada, já que seu propósito original é a renderização gráfica, onde cada pixel de imagem é constantemente atualizado em paralelo (NVIDIA-CUDA^{TM}, 2010). Seu *design* difere primariamente da CPU por dedicar mais transistores ao processamento de dados (ALUs - *Arithmetic Logical Units*) em detrimento do cache de dados e controle de fluxo, como ilustra a Figura 3.2.

Figura 3.2: Diferenças nas arquiteturas da CPU e GPU, retirado de $NVIDIA - CUDA^{TM}, 2010$

Essencialmente, as GPUs do início da década de 2000 foram designadas para produzir

uma certa cor para cada pixel na tela, usando unidades aritméticas programáveis conhecidas como *pixel shaders* (sombreadores de pixel). Em geral, um pixel shader usa a posição (x,y) do pixel na tela, além de informações adicionais - cores, coordenadas de texturas, atributos gráficos - para computar a cor final. Devido ao fato das operações matemáticas realizadas nas cores e texturas de entrada serem completamente controladas pelo programador, pesquisadores concluíram que estas "cores" de entrada poderiam ser na verdade qualquer tipo de dado (Sanders e Kandrot, 2010).

Inicialmente, a programação direcionada à GPU estava restrita a linguagens (APIs) gráficas, como OpenGL e DirectX. Tal situação dificultava bastante a pesquisa em áreas não relacionadas a área gráfica, pois o programador precisava inicialmente converter seus dados para um ambiente gráfico, processá-lo, e por fim reconvertê-lo. Esse processo, além de trabalhoso, poderia apresentar erros inesperados (já que não se estava sendo feita uma computação gráfica em si). A situação mudou em 2006, quando a NVIDIA lançou a GeForce 8800 GTX, a primeira placa gráfica a ser construída com arquitetura CUDA (do inglês, *Compute Unified Device Achitecture*). De forma resumida, esta nova arquitetura (apresentada na seção 2.2) permitia que as ALUs (Unidades Lógicas Aritméticas), antes focadas em processamento de pixels, possuíssem a liberdade de realizar computações de uso geral. Em conjunto com esta nova arquitetura, a NVIDIA criou o CUDA C (e seu compilador nvcc), que nada mais é que a linguagem C com novas bibliotecas e comandos específicos para operar na GPU e CPU (Sanders e Kandrot, 2010).

Quanto às aplicações a GPU é bem adaptada a problemas que podem ser expressos por computação paralela - o mesmo programa é executado em unidades de processamento em paralelo - e com alta intensidade aritmética, que é a razão entre operações aritméticas e operações de memória. Como mostrado, a GPU atual é especializada em cálculos de ponto-flutuante, deixando em segundo plano o controle de fluxo; por esta razão, a GPU tem capacidade inferior de transferência de memória comparada à CPU (NVIDIA-CUDA^{TM}, 2010). Problemas com baixa necessidade de transferência de memória e alta necessidade de cálculos são os que melhor se adequam à programação em GPU, como é o caso da modelagem de tiros e da migração reversa no tempo (ver exemplo na seção 3.3).

3.2 Arquitetura da GPU - CUDA

A unidade gráfica é organizada em sub-unidades de processamento denominadas *threads*. As *threads*, por sua vez, podem ser organizadas em blocos (*blocks*), e os *blocks* podem ser organizados em *grids*, como ilustra a Figura 3.3 (NVIDIA-CUDA^{TM}, 2010).

O número de *grids*, *blocks* por *grid* e *threads* por *block* é definido pelo programador. Cada *thread* tem um identificador único, relacionada à sua posição em certo bloco de certo

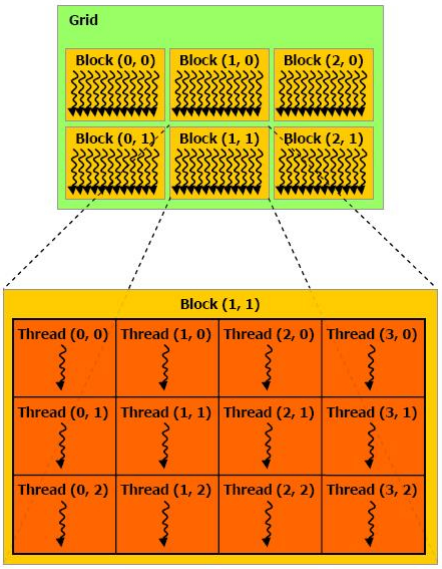


Figura 3.3: Grupos de threads são organizados em blocks; grupos de blocks são organizados em grids, retirado de $NVIDIA - CUDA^{TM}$, 2010

grid. Ao rodar um programa, todas as *threads* solicitadas executam o mesmo código, porém como cada uma tem um identificador único, os resultados são paralelos (no cálculo de um laplaciano em cada ponto de uma malha, por exemplo; a equação é a mesma para todas as *threads*, mas o resultado é distinto devido aos identificadores únicos de cada *thread*).

O processo pode ser ilustrado por um exemplo simples, como a soma de dois vetores de mesma dimensão. O objetivo é somar o vetor 1 (a) com o vetor 2 (b),de forma a obter um vetor 3 (c), onde $a(i) = b(i) + c(i)$:

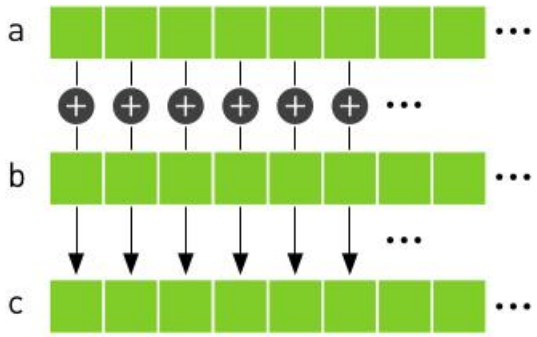


Figura 3.4: Somando dois vetores

Num pseudocódigo serial, o problema pode ser programado da seguinte maneira:

```
Defina a dimensão N dos vetores
Dimensiona a(N), b(N), c(N)
```

```
Preenche a e b; preenche c com zeros
Faça para index de 1 a N {
   c(index) = a(index) + b(index)
}
Imprime c
Fim
```

O seu equivalente em linguagem C seria:

```
#define N 10   //Neste exemplo, N=10
int main() {
  int a[N], b[N], c[N];
  int index;

  //Preenche v1 e v2
  for (int i=0; i<N; i++) {
     a[i] = -i;
     b[i] = i * i;
  }

  //Cálculo da soma
  int index = 0;
  while (index < N) {
    c[index] = a[index] + b[index];
  index += 1;
  }

  //Imprime c
  for (int i=0; i<N; i++) {
     printf( "%d + %d = %d\n", a[i], b[i], c[i] );
  }

return 0;
} //Fim
```

Na etapa "Cálculo da soma", o processador executa N iterações. A reestruturação do código utilizando a GPU permite que esta soma seja feita em apenas 1 iteração, executada por N *threads*. Assim, o código paralelizado, em pseudocódigo seria:

```
Defina a dimensão N dos vetores
```

```
Dimensiona a(N), b(N), c(N) na CPU
Dimensiona a_d(N), b_d(N), c_d(N) na GPU
Preenche a e b; preenche c com zeros na CPU
Copia a, b, c para a_d, b_d, c_d (CPU para GPU)
Execute a função "adição" para N threads, 1 block, 1 grid
Copia c_d para c (GPU para CPU)
Imprime c
Fim

Função adição(a[N], b[N], c[N]) {
  c[index_thread] = a[index_thread] + b[index_thread]
}
```

A diferença fundamental é a variável index_thread, que é única para cada *thread*. Como são chamadas N threads, este índice é preenchido automaticamente: para a primeira *thread* index_thread= 1, para a segunda Thread index_thread= 2, até a *thread* N, onde index_thread= N. Este processo pode ser entendido mais detalhadamente analisando o código em CUDA C (adaptado de Sanders e Kandrot, 2010):

```
#define N 10
int main( void ) {
  int a[N], b[N], c[N];
  int *dev_a, *dev_b, *dev_c;
  // Aloca a memória na GPU
  cudaMalloc( (void**)&dev_a, N * sizeof(int) ) );
  cudaMalloc( (void**)&dev_b, N * sizeof(int) ) );
  cudaMalloc( (void**)&dev_c, N * sizeof(int) ) );
  // Preenche 'a' e 'b' na CPU
  for (int i=0; i<N; i++) {
    a[i] = -i;
    b[i] = i * i;
  }
  //Copia os vetores 'a' e 'b' para a GPU
  cudaMemcpy( (dev_a, a, N*sizeof(int), cudaMemcpyHostToDevice) );
  cudaMemcpy( (dev_b, b, N*sizeof(int), cudaMemcpyHostToDevice) );

  //Cálculo da soma na GPU
  add<<<N,1>>>( dev_a, dev_b, dev_c);

  //Copia 'c' da GPU para a CPU
```

```
  cudaMemcpy( (c, dev_c, N*sizeof(int), cudaMemcpyDeviceToHost ) );

  //Imprime os resultados
  for(int i=0; i<N; i++) {
    printf( "%d + %d = %d\n", a[i], b[i], c[i] );
  }

  //Libera a memória alocada na GPU
  cudaFree (dev_a);
  cudaFree (dev_b);
  cudaFree (dev_c);

  return 0;

}
```

É interessante observar que na etapa "Cálculo da soma na GPU" a definição de N threads, 1 bloco e 1 grid é feita dentro da chamada da função add, entre os delimitadores <<<>>>. Todas as N threads executarão a função add, mas como o identificador *blockIdx.x* é diferente para cada uma, cada *thread* executará uma soma $c(i) = a(i) + b(i)$, reduzindo o número de iterações de N para apenas 1.

A placa com GPU utilizada neste trabalho é formado por uma NVIDIA Tesla C2075. Para os fins propostos, a Tesla C2075 foi suficiente para executar todas as operações. Sua memória é dividida em: global, com 5.636.292.608 *bytes*, e constante, com 65.536 *bytes*, o que resulta em aproximadamente $5Gb$ de memória máxima disponível para cada processamento. A construção de *grids*, *blocks* e *threads* tem os seguintes máximos: 1.024 *threads* por *block*, e 65.535 blocks por *grid*. Utilizando 1 *grid*, é possível então alocar $1.024 \times 65.535 = 67.107.840$ *threads*. Considerando o exemplo dos vetores, poderíamos ter no máximo 3 vetores com 67 milhões de posições cada; em todos os testes realizados com modelos 2D a memória da GPU foi suficiente, não sendo necessário armazenar dados em disco no decorrer do processo.

Como exemplo, analisamos a memória utilizada a cada tiro durante a migração reversa no tempo do modelo Marmousi, disponibilizado pelo IFP:

- O modelo de velocidades tem dimensões 369×375. Sua alocação na GPU requer $|VEL| = 369 \times 375 = 138.375$ posições.
- Na geometria, são utilizados 96 canais por tiro.
- Com base na Figura 2.2 da seção 2, alocamos a malha. São 96 posições relativas aos canais; mais 96 posições relativas à expansão lateral; mais 60 posições relativas à borda

absorvedora. Assim, $nx = 96 + 96 + 60 = 252$.

- Para nz, é utilizada toda a extensão vertical do modelo mais a borda. $nz = 375 + 60 = 435$.
- A matriz $MOD(x, z, t)$, que conterá os *snapshots* do campo de pressões da modelagem, tem dimensões $nx * nz * nt$, neste caso, $|MOD| = 252 * 435 * 725 = 79.474.500$, pois este dado tem 725 amostras temporais por traço sísmico.
- Na migração reversa no tempo, a imagem vai sendo construída ao mesmo tempo em que o dado do tiro é depropagado. Assim, só é necessário que se aloque a matriz do campo depropagado $SHOT(x, z, t)$, e $SHOT_{aux}(x, z, t)$ que armazena o campo no instante $(t + \Delta t)$ (ver equação 1.11). Assim, são necessárias $2 * nx * nz$ posições: $|SHOT| = 2 * 435 * 252 = 219.240$ posições.
- Desprezando a memória necessária para salvar as fatias da imagem durante a condição de imagem (visto que elas são transferidas para a GPU a cada iteração, liberando a memória utilizada), no total são alocadas $|MOD| + |SHOT| + |VEL| = 79474500 + 219240 + 138375 = 79.832.115$ posições. Como cada pressão do campo de onda (e cada valor de velocidade do campo de velocidades) é um número real em ponto flutuante, para calcular o número total de *bytes* multiplicamos por 4. Assim, $MEM = 79832115 * 4 = 319.328.460$ *bytes*.
- Sabe-se que para a placa utilizada neste trabalho, a memória máxima da GPU é de $MEMGPU = 5.636.292.608$ *bytes*. Calculando a razão entre as memórias, $\frac{MEM}{MEMGPU} \approx 0,0567$, ou seja, para cada tiro na migração reversa no tempo para este modelo 2D é utilizada aproximadamente 6% da memória da GPU.

Estendendo este resultado, no qual só são utilizados 6% da memória para o modelo Marmousi, pode-se concluir que seria possível realizar testes em GPU com modelos 2D duas vezes mais profundos, ou com 2 vezes mais amostras por traço sísmico, ou até mesmo ambos, com base nas proporções da análise anterior. No caso de um modelo 3D, seria introduzida uma nova dimensão, e um novo fator multiplicativo em $|MEM|$ provavelmente maior do que 100 amostras; nesta situação a memória da NVIDIA Tesla C2075 seria insuficiente para migrar um tiro deste modelo. Nesse caso, a ser estudado em trabalhos futuros, provavelmente seria necessária uma das técnicas citadas ao final do Capítulo 1, para reduzir a memória necessária da matriz (que neste caso possuiria dimensão y) de *snapshots* $MOD(x, y, z, t)$.

3.3 Exemplo de migração reversa no tempo implementada em GPU

Nesta seção revisitamos o fluxograma para a RTM apresentado na seção 2.2 do Capítulo 2, modificando-o para implementação em GPU. No fluxograma abaixo são postas em negrito

as modificações decorrentes da implementação em GPU:

1. Acessa o *header* do dado de tiros e obtém as posições dos geofones e fonte para o tiro atual. Copia o campo de velocidades para o programa e aloca a malha onde será propagado o campo de pressões **na GPU**, similar à malha **3** da Figura 2.1, com base na geometria do tiro atual;

2. Faz uma modelagem sintética com os mesmos parâmetros (posições de geofone e fonte) do tiro atual. Começa as iterações no tempo (de $t = 0$ a $t = t_{max}$):

   - Insere a fonte em sua posição nos tempos iniciais;
   - Propaga o campo de onda pelo modelo, aplicando a equação do REM em cada ponto da malha, que **corresponde a uma *thread* individual**;
   - Grava numa matriz 3D o estado atual do campo de pressões, $MOD(x, z, t_{atual})$. Ao final a matriz conterá todos os snaps desde $t = 0$ a $t = t_{max}$, ou seja, $MOD(x, z, t)$;

3. Inicia a migração reversa no tempo. Começa um novo *loop* de iterações temporais, partindo de t_{max} até $t = 0$:

   - Aloca uma nova malha **na GPU**, com as mesmas dimensões. Insere os valores de $P(x, z, t = t_{max})$ provenientes do tiro atual;
   - Aplica o REM depropagando no tempo, **em cada thread**, e salva o snapshot em $SHOT(x, z, t_{max} - \Delta t)$;
   - Aplica a condição de imagem (**em cada thread**):
     $I_j(x, z) = I_{j-1}(x, z) + SHOT(x, z, t_{max} - j\Delta t) * MOD(x, z, t_{max} - j\Delta t)$
     A imagem $I_j(x, z)$ é atualizada a cada iteração j, de forma que ao chegar em $t = 0$ é obtida a imagem completa para o tiro atual.

4. Volta à etapa 1 com o próximo tiro, e repete o processo. Finaliza ao chegar no último tiro.

Na migração reversa no tempo, cada tiro é processado em uma série de passos que representam avanço ou retrocesso no tempo. O número total de iterações no tempo, n_{it}, é dado por:

$$n_{it} = 2 * nshots * nt. \tag{3.1}$$

Sendo $nshots$ o número total de tiros e nt o número de amostras temporais por traço sísmico. Assim, num único processador, a cada tiro, são realizadas nt iterações temporais para o

avanço temporal e mais nt iterações para o retrocesso no tempo. A cada iteração é atualizado o campo de pressões $P(x, z, t + \Delta t)$ (ou $P(x, z, t - \Delta t)$ no retrocesso temporal) para cada ponto da malha, ou seja, $nx \times nz$ pontos. A diferença fundamental é que no código em GPU são alocadas exatamente $nx * nz$ threads (uma para cada ponto da malha), de forma que cada ponto executa o conjunto de operações associado ao cálculo do campo extrapolado (equação 1.11) em paralelo, a cada iteração. Desta forma, o tempo de execução é reduzido em um fator proporcional a $nx \times nz$, neste caso, comparando o código serial e o código em GPU.

No trabalho de dos Santos (2010) foi processado o dado do modelo Marmousi, de domínio público (disponibilizado pelo *Institute Français du Pétrole*), com uso do REM, em série e num código paralelizado. Este dado de domínio público do modelo Marmousi tem os seguintes parâmetros: 240 tiros, 96 receptores por tiro; 725 amostras por traço, intervalo de amostragem $4ms$; dimensões do modelo de velocidades 369×375, com amostragem espacial de $25m$ e $8m$. O processamento deste dado num código serial, utilizando o REM, levou $7h12min$. O mesmo dado migrado em código paralelizado, não em GPU, mas em 120 processadores, levou um tempo de aproximadamente $6min$. Na implementação deste trabalho, a migração do dado Marmousi de domínio público, com estes mesmos parâmetros, levou um tempo aproximado de $5min53s$, mostrando consistência entre os resultados de ambos os trabalhos. Assim, a redução de tempo proporcionada pela GPU se deve a alocação de $nx \times nz$ threads por tiro, de forma que os cálculos de cada ponto da malha, a cada iteraçao temporal, são realizados simultaneamente (em paralelo). O ganho temporal foi consistente com aquele obtido por dos Santos (2010), de 72,8 vezes ($7h12min$ para $5min53s$); a qualidade da imagem final do processo serial se mostrou idêntica (ver Capítulo "Resultados") à dos processos em paralelo, tanto em vários processadores (dos Santos, 2010) quanto em GPU.

CAPÍTULO 4

Resultados

Nesta seção são apresentados resultados para a migração reversa no tempo. Foram utilizados dois modelos: o modelo Marmousi e o modelo denominado "Falhas". As seções seguintes mostram os resultados para cada modelo.

4.1 Modelo Marmousi

O modelo Marmousi foi criado pelo *Institute Français du Pétrole* em 1988. A geometria deste modelo é baseada num perfil que corta a bacia de Cuanza e a parte Norte de Quenguela, em Angola. A geometria e modelo de velocidades foram criados de forma a produzir dados sísmicos complexos, que requerem técnicas avançadas de processamento para obter uma imagem correta da subsuperfície. O modelo de velocidades tem $9,225km$ de extensão e $3km$ de profundidade, correspondendo a 369×375 amostras espaciais, com espaçamentos de $25m$ e $8m$ para x e z (direção horizontal e vertical, respectivamente).

O modelo Marmousi contém 158 camadas horizontais, cortadas por diversas falhas normais, resultando numa série de blocos deslocados que tornam o modelo complicado em sua parte central; possui também cunhas salinas na parte mais profunda do modelo, marcando uma discordância entre o sistema de falhas normais (acima) e um sistema de dobramentos (abaixo), no qual se encontra o possível reservatório de hidrocarbonetos, na parte mais alta do sinclinal (Figura 4.1). Assim, a maior dificuldade em imagear o modelo é devido à alta variação de velocidade lateral; outra dificuldade se deve ao alto contraste de impedância da cunha salina com o modelo, de forma que quase toda energia do campo de onda é refletida e pequena parte é transmitida, dificultando o imageamento do sistema de dobras logo abaixo destas cunhas salinas.

A migração reversa no tempo pré-empilhamento consegue imagear bem mesmo com variações laterais de velocidade, por usar uma solução da equação completa da onda acústica, considerando todas as direções do campo de onda. É também eficiente ao imagear as estruturas abaixo das cunhas salinas, em parte por conseguir construir os refletores utilizando o caminho das ondas prismáticas.

De acordo com a descrição do modelo Marmousi dado por Irons (1988), os dados de tiro disponibilizados ao público foram gerados por algoritmos de diferenças finitas utilizando o pacote Madagascar. O primeiro resultado corresponde à migração destes dados. Seus principais parâmetros, com base em seu header, são:

- Arranjo *end-on* de 96 geofones com fonte à direita;
- Offset mínimo de $200m$;
- Espaçamento entre geofones de $25m$;
- A fonte do primeiro tiro é disparada em $x = 3000m$, e o primeiro geofone do arranjo em $x = 400m$ ($2600m$ é o offset máximo);
- Número total de tiros de 240, com $25m$ de espaçamento entre tiros;
- 725 amostras por traço, intervalo temporal de $4ms$;
- Frequência dominante de $20Hz$, com base no espectro de frequência;
- Total de 23401 traços.

Na migração reversa no tempo para este dado utilizou-se como fonte uma wavelet Ricker de frequência máxima $20Hz$. O tempo total de migração foi de $353,6s$ ou $5min54s$; como mostrado no Capítulo 3, dos Santos (2010) obteve resultados em um único processador para este modelo com parâmetros similares, também utilizando o método de expansão rápida para extrapolar os campos de onda, num tempo de $7h15min$; a comparação entre os resultados em série do trabalho de W.G. dos Santos e os resultados deste trabalho mostram que houve um ganho de $72,3$ vezes na velocidade de processamento. O resultado da RTM para esse dado de tiros disponibilizado pelo IFP é mostrado na Figura 4.2.

Posteriormente, foi feita uma modelagem de tiros do dado Marmousi, utilizando os mesmos parâmetros do *header* do dado de domínio público. O tempo total de modelagem foi de $236.34s$, ou $3min54s$. A migração levou o mesmo tempo que para o dado de domínio público ($5min54s$). O resultado é mostrado na Figura 4.3. Como é possível notar, os resultados são inteiramente similares para as Figuras 4.2 e 4.3, mostrando que o algoritmo implementado neste trabalho reconstitui o dado de domínio público de forma fiel.

Subsequentemente, foi repetido o processo de modelagem sintética e migração reversa no tempo, com os mesmos parâmetros, exceto a frequência máxima da wavelet onde ao invés de utilizar $20Hz$ foi usado $35Hz$. Só foi possível testar esta nova frequência sem a necessidade de reamostrar o dado (requerido no caso da aproximação por operador de diferenças finitas de segunda ordem no tempo) devido à implementação pelo REM, que permite o cálculo do campo de pressões em amostragens diferentes através de uma nova escolha do número

máximo de termos M da expansão da função cosseno (equação 1.12). Como esperado, uma maior frequência máxima gera uma melhor resolução; nesta imagem (Figura 4.4) os refletores estão melhor delineados, especialmente os flancos de sal, na profundidade de $2,5km$ e também o sinclinal que contém um suposto reservatório de hidrocarbonetos, localizado em $z = 2,7km$ e $x = 7,5km$.

Por último, optou-se por gerar um dado sintético utilizando um arranjo diferente cobrindo todo o modelo (já que o dado de domínio público o primeiro geofone do primeiro tiro se localiza a $x = 425m$, daí a má qualidade da imagem antes de $425m$). Os parâmetros utilizados foram:

- Arranjo splid-spread variável com 180 geofones; se inicia *end-on* com a fonte na esquerda, e à medida que se move torna-se splid-spread. Ao alcançar a marca de $180 \times 25 = 4500m$, torna-se *split-spread* simétrico, com 90 geofones de cada lado da fonte. À medida que se aproxima do lado direito do modelo, o processo é análogo, tornando-se *split-spread* assimétrico e por fim *end-on* com fonte na direita, no último tiro.
- Foram modelados o máximo de tiros possível, 369, já que o modelo Marmousi possui 369 amostras horizontais.
- 725 amostras por traço, $\Delta t = 4ms$;
- Wavelet Ricker de 35Hz.

A modelagem dos 369 tiros levou $455,2s$ (ou $7min36s$), e sua migração $643s$ ($10min43s$). Assumindo a mesma proporção de tempo ($6min$ correspondem a $7h15m$), é possível estimar que os $17m19s$ (modelagem e RTM) em GPU levariam aproximadamente $19h$ em um só processador. Esta análise reforça a importância da paralelização, que permite verificar em pouco tempo o processamento de uma grande quantidade de tiros. O resultado pode ser visto na Figura 4.5. A diferença mais notável em relação à migração da modelagem *end-on* é que a área do modelo anterior a $x = 400m$ foi bem melhor imageada (destaque para os refletores inclinados logo abaixo da cunha de sal à esquerda, melhor definidos que na Figura 4.4). Percebe-se também que a migração do dado *split-spread* conseguiu imagear melhor as feições da lateral direita do modelo, como o refletor em $x = 8,8km$, $z = 1,8km$, que aparece quebrado na Figura 4.3 e bem definido na Figura 4.4; o mesmo ocorre no canto inferior direito do modelo, melhor delineado na Figura 4.5.

4.1.1 Modelagem sísmica para o modelo Marmousi

Foram realizadas três migrações reversas no tempo para o modelo Marmousi. Em duas dessas migrações, foi necessário modelar os tiros a partir do campo de velocidades. A primeira

delas foi a geração de um dado sísmico com geometria igual ao dado de tiros para o Marmo disponibilizado pelo IFP. Na Figura 4.6 são mostrados os primeiros três tiros para o dado original (Figura 4.6(a)), e para o dado modelado (Figura 4.6(b)). O arranjo de geofones é *end-on*, com fonte à direita. Foram modelados 240 tiros, com 96 canais cada, 725 amostras temporais por traço, $\Delta t = 4ms$ e $f_{max} = 20Hz$. Na segunda migração, foram modelados também 240 tiros com os mesmos parâmetros previamente citados, exceto a frequência máxima, que foi de $35Hz$ neste caso. Ambas as modelagens foram realizadas em algoritmos implementados em GPU e levaram um tempo de execução de aproximadamente $5min54$.

O terceiro teste foi a migração dos 369 tiros com arranjo *split-spread* variável (Figura 4.7). Foram modelados 369 tiros com 120 canais cada, num arranjo *split-spread* variável ao longo dos tiros: no primeiro tiro o arranjo é *end-on*, com fonte na direita; no segundo, há 1 geofone à esquerda da fonte e 119 à direita; no terceiro, 2 à esquerda e 118 à direita (Figura 4.7(a)), e assim por diante, até ficar *split-spread* simétrico (Figura 4.7(b)); conforme o arranjo se aproxima do final do modelo, se torna *split-spread* assimétrico novamente, e no último tiro *end-on* com fonte na direita (Figura 4.7(c)). Esta modelagem teve um tempo de execução de $7min36s$.

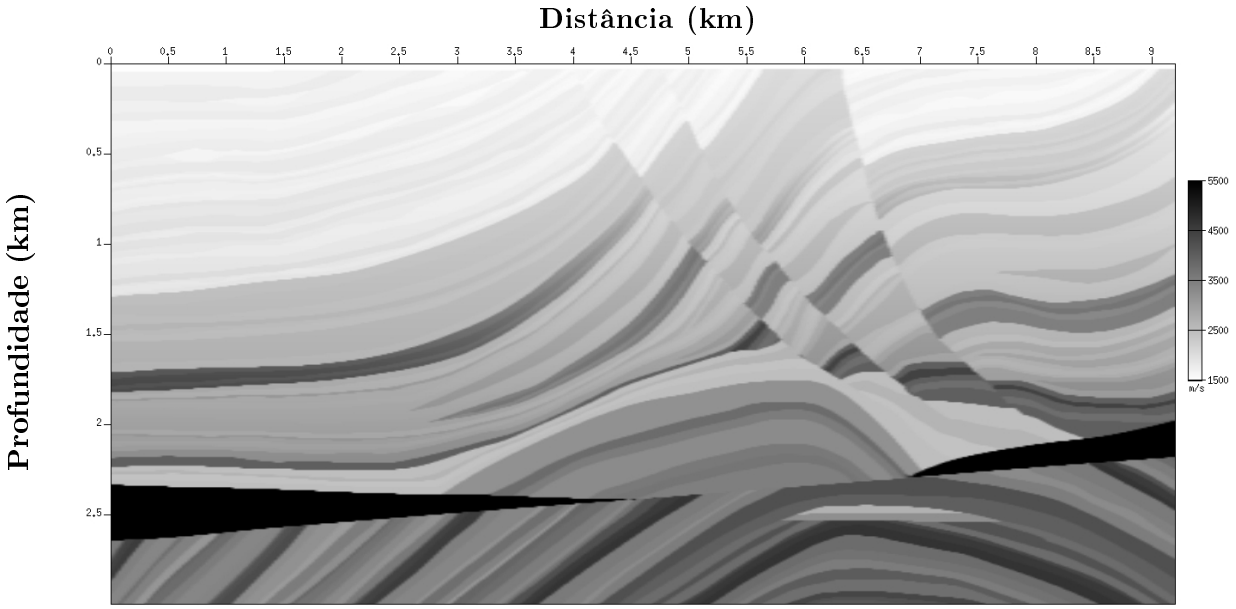


Figura 4.1: Modelo de velocidades Marmousi.

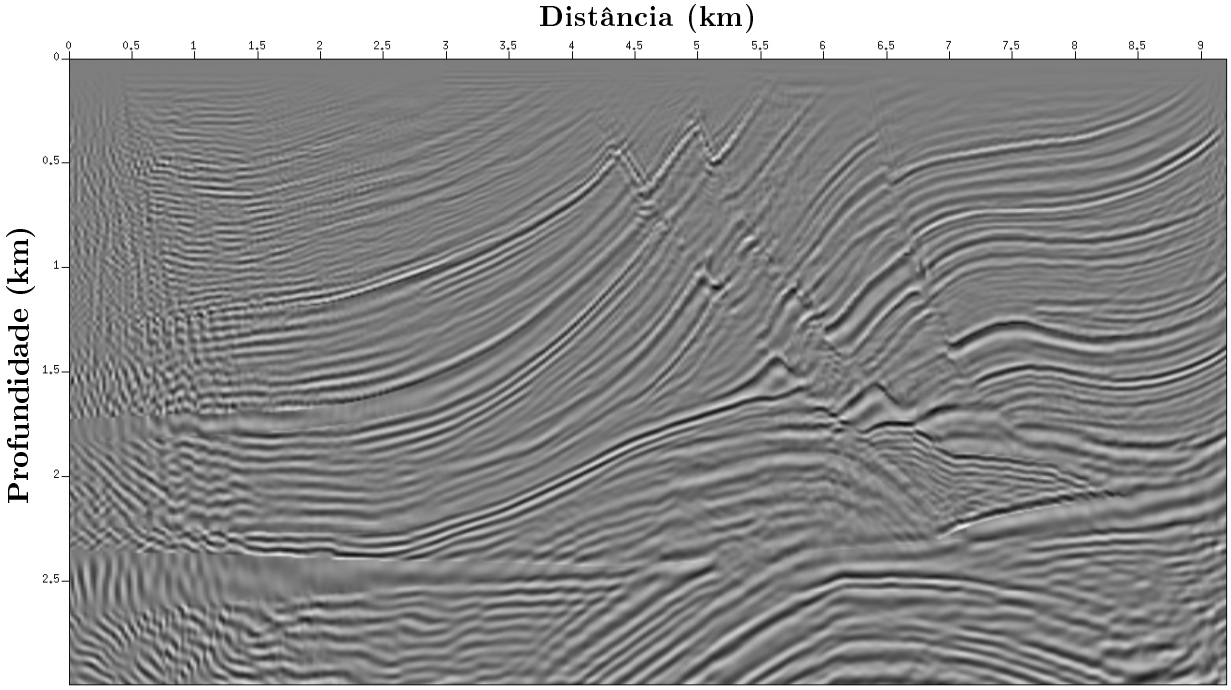


Figura 4.2: Migração dos tiros de domínio público, utilizando $f_{max} = 20Hz$, para o modelo Marmousi.

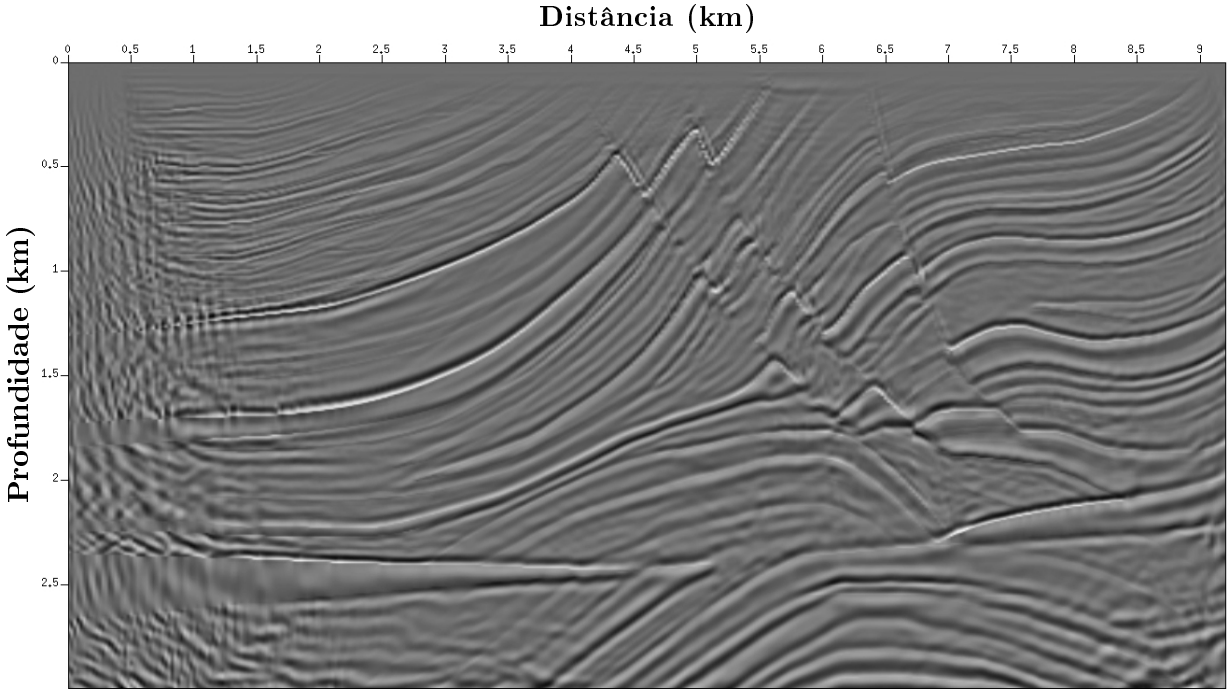


Figura 4.3: Migração dos tiros modelados sinteticamente, utilizando $f_{max} = 20Hz$, para o modelo Marmousi.

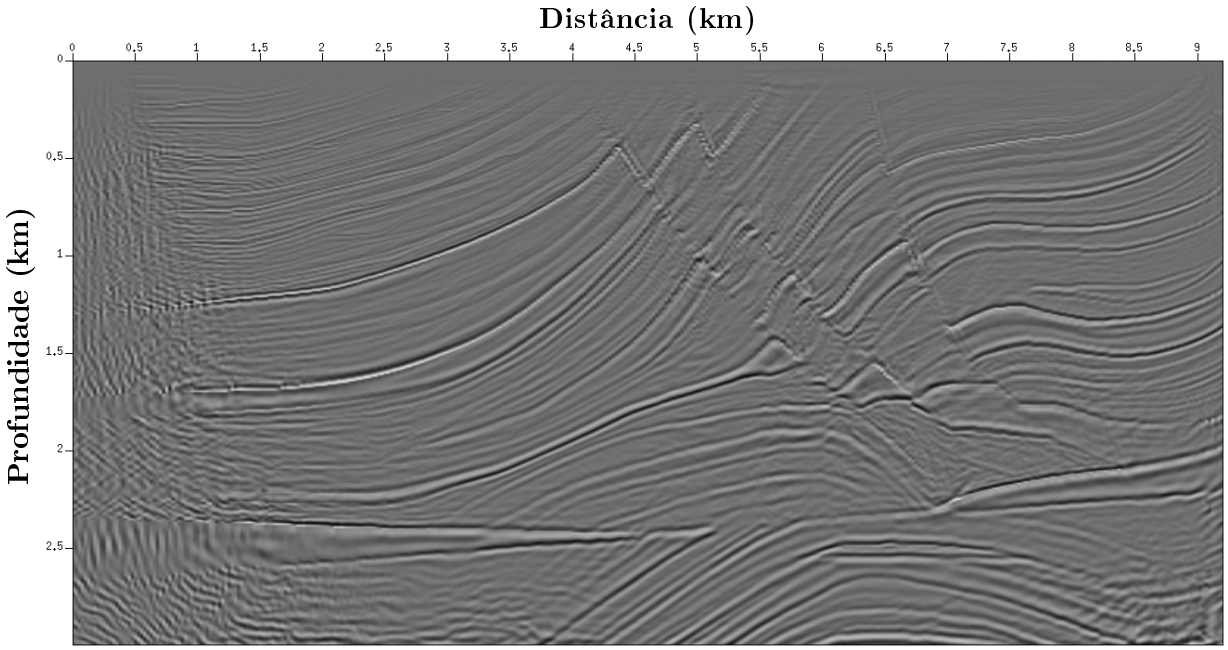


Figura 4.4: Migração dos tiros modelados sinteticamente, utilizando $f_{max} = 35Hz$, para o modelo Marmousi.

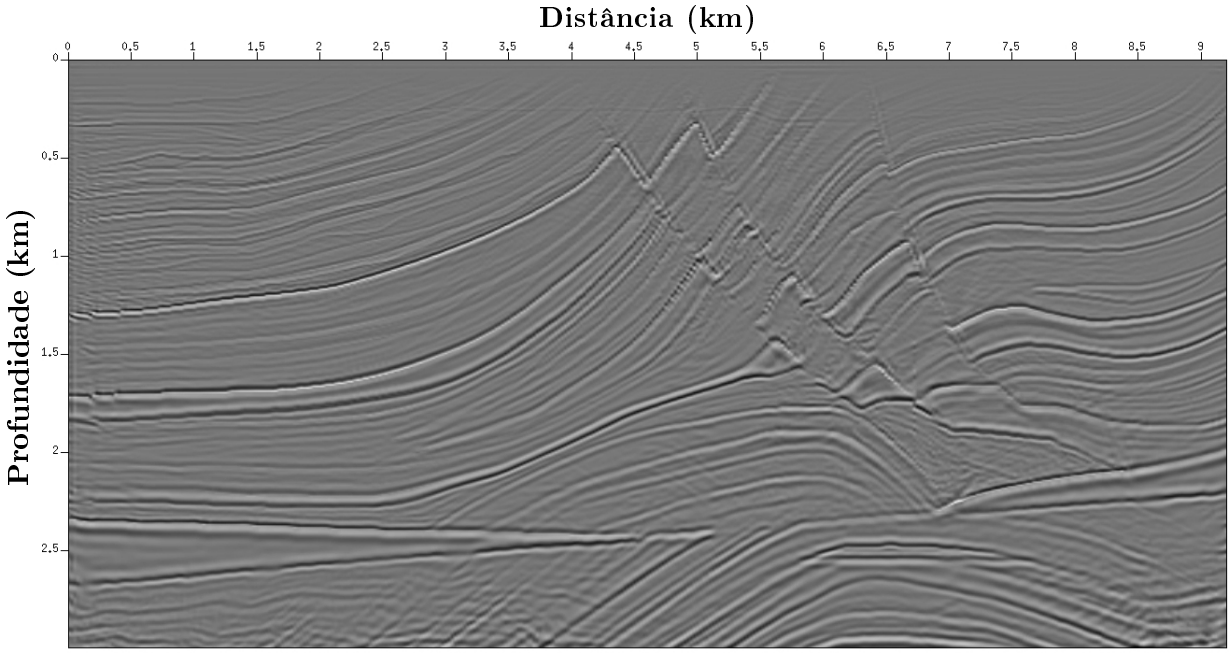


Figura 4.5: Migração do dado sintético de arranjo *split-spread* assimétrico e 369 tiros, utilizando $f_{max} = 35Hz$, para o modelo Marmousi.

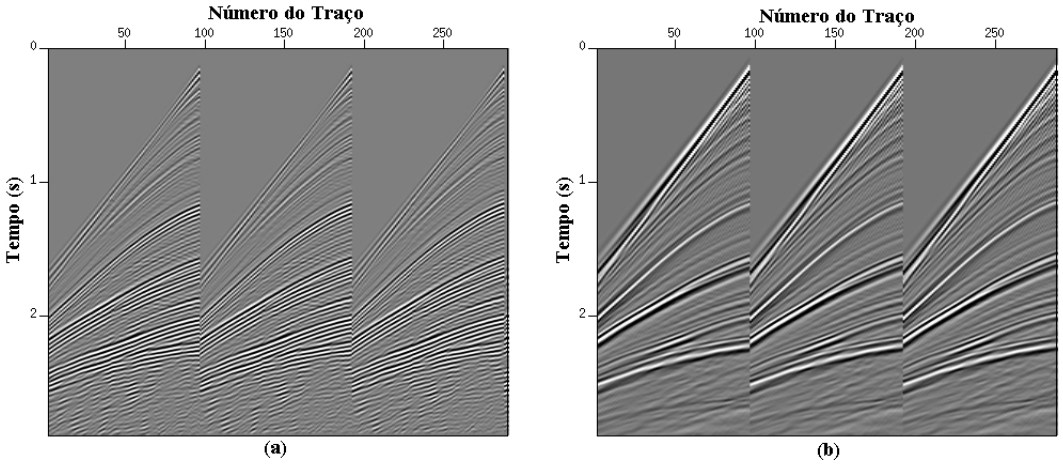


Figura 4.6: Primeiros três tiros do dado Marmousi, arranjo end-on com fonte à direita. Dado original em (a), dado modelado em (b).

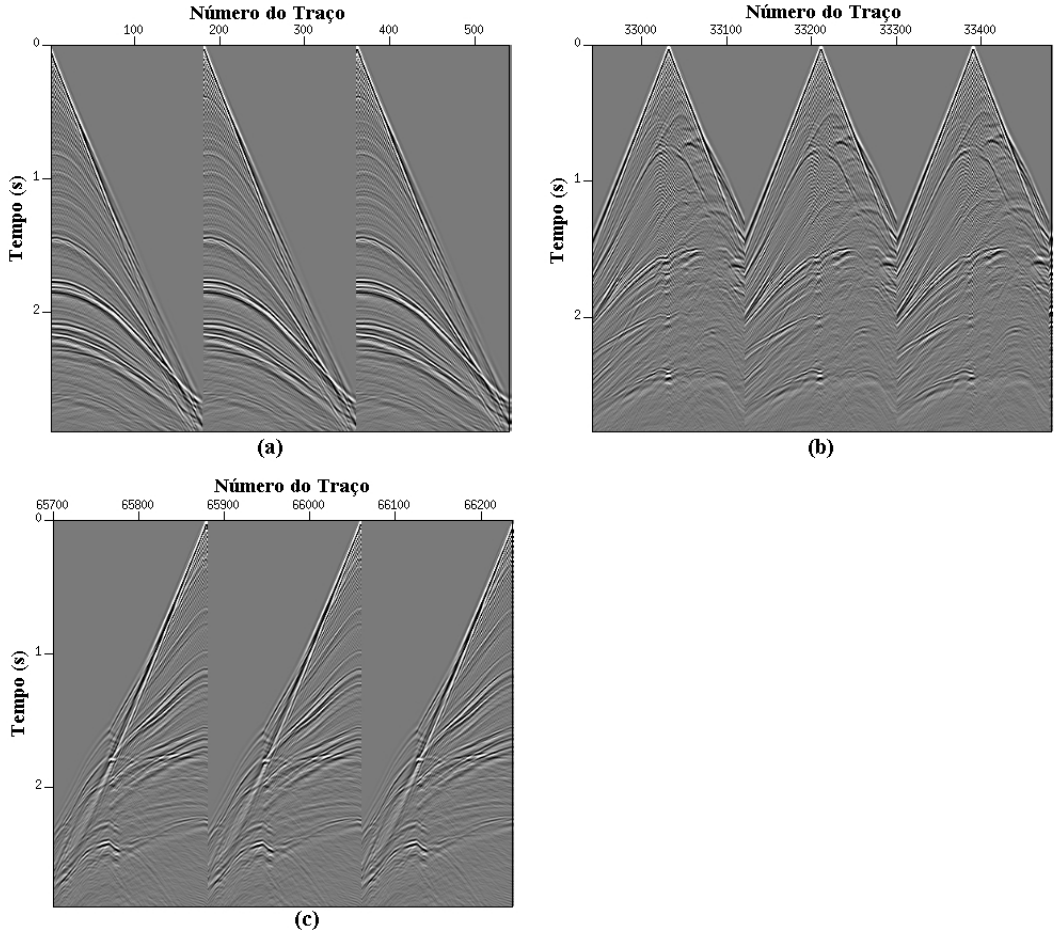


Figura 4.7: Alguns tiros do dado modelado de arranjo *split-spread* variável, modelo Marmousi. O dado se inicia *end-on* (a), torna-se *split-spread* (b), e volta a ser *end-on* no final do modelo (c).

4.2 Modelo Falhas

O algoritmo foi testado também para outro modelo de grande complexidade geológica, denominado "Falhas" (Figura 4.8). Este modelo tem dimensões de 600 amostras horizontais e 265 amostras verticais; os intervalos espaciais utilizados foram $\Delta x = \Delta z = 10m$, resultando num modelo final de $6km$ por $2.65km$.

Foram utilizados os seguintes parâmetros, na modelagem e migração:

- Arranjo *split-spread* variável de 120 geofones (inicia *end-on* com fonte na esquerda, depois *split-spread* assimétrico, *split-spread* simétrico, *split-spread* assimétrico novamente, por fim *end-on* com fonte na direita);
- *Wavelet* Ricker de frequência máxima 35 Hz;
- 725 amostras temporais por traço, $\Delta t = 4ms$;
- 600 tiros no total.

O tempo de modelagem foi de $448.1s$ (ou $7min28s$) e o de migração foi $631.7s$ (ou $10min32s$). A Figura 4.9 mostra o resultado da migração. Apesar de ser um modelo bastante complexo, a maioria das falhas foi delineada na imagem sísmica. O diápiro em $x = 4km, z = 2km$ foi bem imageado; os demais diápiros desta camada nem tanto, o que é atribuído aos pontos difratores de menor velocidade internos a esta camada. Estes pontos são bem imageados, mas parece que a energia da difração dos dois pontos mais altos contamina a delineação da camada, que acaba sendo imageada erroneamente como um refletor plano. Percebe-se também que os dobramentos internos à camada dos diápiros foram bem resolvidos na marca $x = 1,5km, z = 2,2km$, e ligeiramente delineados nas demais áreas.

4.2.1 Modelagem sísmica para o modelo Falhas

Para o modelo Falhas, não havia disponibilizado ao público um dado de tiros. Neste trabalho foi modelado em GPU um dado com 600 tiros, com 120 canais, 725 amostras temporais por traço, $\Delta t = 4ms$, $f_{max} = 35Hz$ e de arranjo *split-spread* variável (como explicado na seção de modelagem para o modelo Marmousi). Alguns tiros são mostrados na Figura 4.10. A modelagem desse dado levou um tempo de execução de $7min28s$.

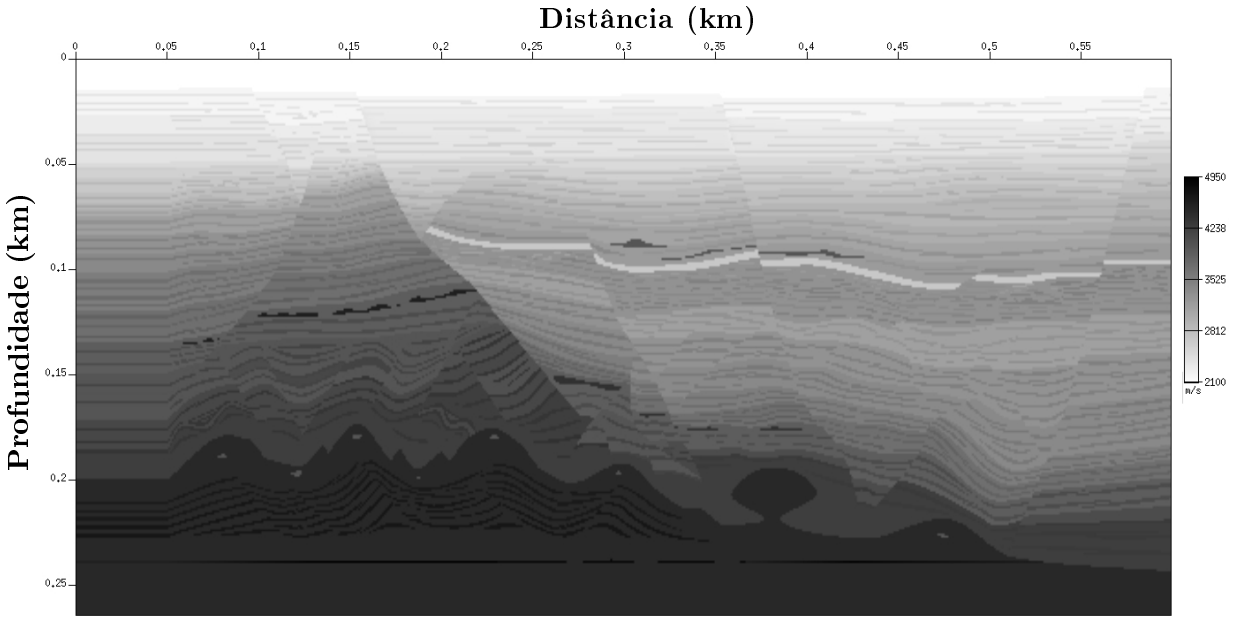


Figura 4.8: Modelo de velocidade "Falhas".

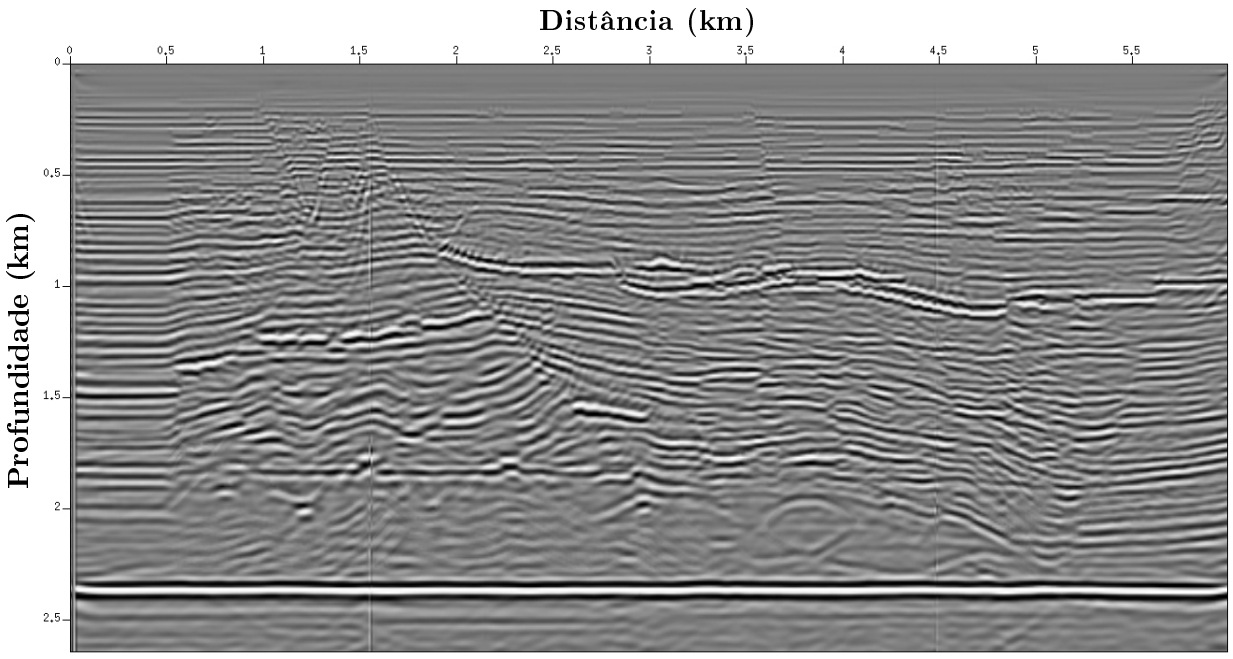


Figura 4.9: Seção migrada correspondente ao modelo Falhas.

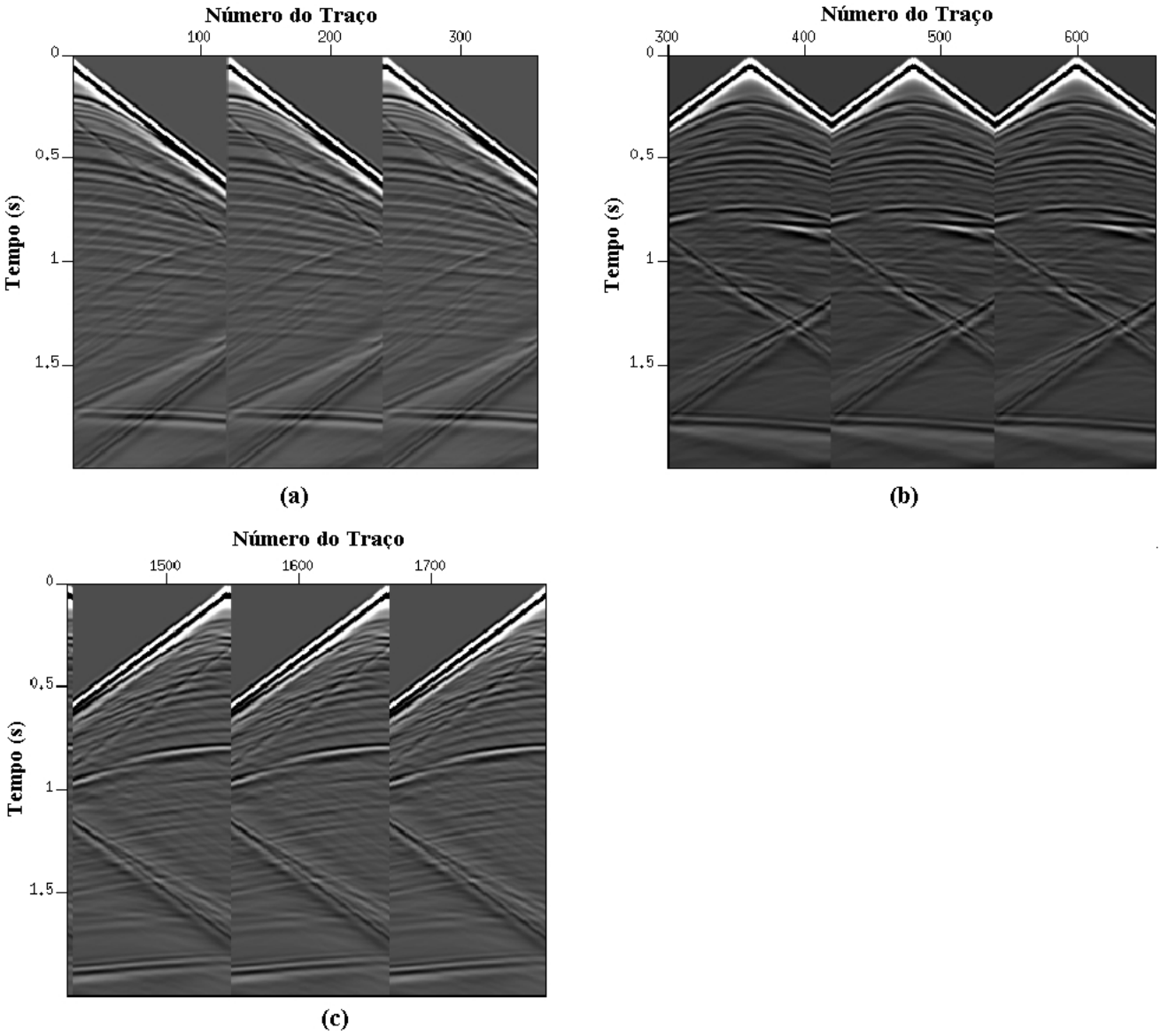


Figura 4.10: Alguns tiros do dado modelado de arranjo *split-spread* variável, modelo Falhas.

CAPÍTULO 5

Conclusões

Neste trabalho foi possível comprovar a eficácia dos algoritmos implementados em GPU, tanto para modelagem sísmica quanto para a migração reversa no tempo. No caso do modelo de velocidades Marmousi, foi possível gerar um dado sintético cujo resultado da seção migrada se mostrou idêntico ao dado público disponibilizado pelo IFP. Este dado, de 240 tiros, modelado por algoritmos de diferenças finitas, foi remodelado de forma a melhorar ao máximo a imagem, visto que no dado disponibilizado a aquisição se iniciava com o primeiro geofone em $425m$, gerando uma imagem ruim antes desta marca; desta forma foram modelados 369 tiros com arranjo split-spread variável de 180 geofones, e a frequência aumentada (em relação ao dado disponibilizado) de $20Hz$ para $35Hz$. Como resultado foi obtida uma seção migrada para o modelo Marmousi imageando todo o modelo de velocidades (especialmente a marca antes dos $425m$) e delineando melhor os refletores.

Os resultados deste trabalho estão consistentes com os resultados do trabalho de dos Santos, 2010, que implementou RTM via REM em algoritmos em série e paralelo (diversos processadores), e cujo ganho de tempo foi de $7h15min$ em série para aproximadamente $6min$ em paralelo (120 processadores) para o modelo Marmousi disponibilizado pelo IFP. Neste trabalho, para este mesmo modelo, o processo de migração no código em GPU foi de $5min53$ (ganho muito próximo comparado aos $6min$ da implementação de dos Santos, 2010).

Resultados análogos foram obtidos para o outro modelo 2D de velocidades testado, o modelo "Falhas". De grande complexidade, dimensões espaciais 600×250, foram modelados 600 tiros, com 120 canais por tiro (também visando a melhor resolução possível) em $7min28s$; a migração reversa no tempo durou $10min32s$. Foi possível notar na imagem final a maioria das falhas, pontos difratores internos aos diápiros, feições internas aos diápiros, além de diversas estruturas isoladas que puderam ser bem imageadas.

Desta forma, é possível concluir que os testes realizados neste trabalho, para modelos 2D de dimensões 369×375 (o Marmousi) e 600×250 (o Falhas), utilizando 725 amostras temporais por traço e $\Delta t = 4ms$ puderam ser realizados em 11 minutos ou menos, onde houve memória possível para armazenar todos os instantâneos do campo de onda do avanço temporal, na migração reversa no tempo (ou seja, não foram necessárias técnicas para armazenar o campo de onda e recuperá-lo, i.e. *checkpointing*, pois a memória da GPU foi

suficiente para armazenar os snapshots, dadas as dimensões dos campos 2D testados).

Foi também desenvolvido um algoritmo de modelagem sísmica que pode ser executado indepentemente da migração reversa no tempo, o qual para os 2 modelos, cujas dimensões e parâmetros foram previamente citados, levou um tempo de execução de no máximo 8 minutos (no caso de 600 tiros para o modelo Falhas). Este algoritmo também permite gerar arquivos de *snapshots*, que pode ter diversos propósitos, como por exemplo visualizar a eficácia de certa condição de borda absorvedora, ou mesmo analisar a forma do campo de onda conforme este se propaga no modelo, seja em avanço ou em retrocesso temporal.

A utilização do método de expansão rápida permitiu que as extrapolações do campo de onda diretas e reversas no tempo fossem realizadas com estabilidade numérica garantida, pois qualquer que fosse o Δt utilizado a convergência da expansão cosseno é garantida escolhendo um número de termos M da aproximação tal que $M > R\Delta t$.

Desta forma, comprovada as vantagens e a eficácia destes algoritmos de modelagem e migração reversa no tempo, em GPU, para modelos 2D, espera-se que em trabalhos futuros estes módulos possam ser adaptados a métodos de imageamento mais robustos, como migração mínimos quadrados (*Least Squares Migration*, LSM) ou inversão completa da forma de onda (*Full Waveform Inversion*, FWI). Outra possibilidade futura é o imageamento de modelos sísmicos em três dimensões, no qual, relembrando a análise de memória da unidade gráfica de processamento NVIDIA Tesla C2075, seria necessária adaptação do código para reduzir o custo de memória da matriz de snapshots do campo modelado, através de técnicas similares às citadas no final do Capítulo 1.

Agradecimentos

Agradeço em primeiro lugar ao professor Dr. Reynam Pestana pela oportunidade de trabalhar neste projeto, pelos ensinamentos teóricos e pelo senso de pragmatismo e produtividade que absorvi no decorrer do trabalho. Agradeço também a Adriano Wagner, que iniciou junto ao professor Reynam Pestana o desenvolvimento de algoritmos em GPU nesta área tão interessante da geofísica, e cujo trabalho tive o prazer de dar continuidade. Ambos foram fundamentais à realização deste trabalho final de graduação.

Aos amigos que me mantiveram motivado a sempre compreender mais profundamente os conceitos geofísicos, Josimar Roberto, Vinicius Carneiro, Laian Silva.

Agradeço também a ajuda de Peterson Nogueira, Rygmary Valera, Rodrigo Santana e Dr. Hedison Sato pelo auxílio no decorrer do trabalho.

Em especial aos professores Milton Porsani, Wilson Figueiró e Michelangelo Gomes, pelo auxílio nas dúvidas durante todo o curso.

À ajuda financeira do PRH08-ANP e CNPq no decorrer do curso.

Ao corpo docente do CPGG-UFBA pelos ensinamentos.

Por fim, à minha família e amigos pelo apoio durante todo o curso.

Referências Bibliográficas

Araújo, E. (2009) Análise dos Métodos de Diferenças Finitas e Expansão Rápida na Migração Reversa no Tempo, Dissertação de Mestrado, Universidade Federal da Bahia, Salvador, BA.

Baysal, E.; Kosloff, D. e Sherwood, J. (1983a) Reverse time migration, Geophysics, **44**:1514–1524.

Dussaud, E.; Symmes, W.; Williamson, P.; Lemaistre, L. e Singer, P. (2008) Computational strategies for reverse-time migration, 78th Annual International Meeting, SEG, Expanded Abstracts, p. 2267¿2271.

Irons, T. (1988) Marmousi Model, 52nd eaeg meeting, Institut Français du Pétrole, Rueil-Malmaison, Paris.

Levin, S. (1984) Principle of reverse-time migration, Geophysics, **49**:581–583.

Lines, R.; Slawinski, R. e Bording, P. (1999) A recipe for stability analysis of finite-difference wave-equation computations, Geophysics, **64**:967–969.

Liu, F.; Zhang, G.; Morton, A. e Leveille, J. (2007) Reverse time migration using one-way wavefield imaging condition, 77th Annual International Meeting, SEG, Expanded Abstracts, pp. 2170–2175.

Liu, F.; Zhang, G.; Morton, A. e Leveille, J. (2011) An effective imaging condition for reverse-time migration using wavefield decomposition, Geophysics, **76**:S29–S39.

Loewenthal, D. e Mufti, I. (1983) Reversed time migration in spatial frequency domain, Geophysics, **48**:627–635.

McMechan, G. A. (1983) Migration by extrapolation of time-dependent boundary values, Geophysical Prospecting, **31**:413¿420.

NVIDIA-CUDA^{TM} (2010) NVIDIA CUDA C Programming Guide, Version 3.1.1, Santa Clara, CA.

Pestana, R. e Stoffa, P. (2010) Time evolution of the wave equation using rapid expansion method, Geophysics, **75**:121–131.

Sanders, J. e Kandrot, E. (2010) CUDA by example, Addison-Wesley, Upper Saddle River, NJ.

dos Santos, A. (2010) Desafios computacionais da Migração Reversa no Tempo Pré-empilhamento, Trabalho de Graduação, Universidade Federal da Bahia, Salvador-BA.

dos Santos, A. (2013) Inversão de forma de onda aplicada à análise de velocidades sísmicas utilizando uma abordagem multiescala, Disseração de Mestrado, Universidade Federal da Bahia, Salvador-BA.

Symes, W. (2007) Reverse-time migration with optimal check pointing, Geophysics, **72**:SM213–SM221.

Whitmore, D. N. (1983) Iterative depth imaging by back time propagation, 53rd Annual International Meeting, SEG, Expanded Abstracts, p. 382¿385.

Yilmaz, O. (2001) Seismic Data Analysis, Society of Exploration Geophysicists, Tulsa, OK, USA.