Análise de um algoritmo de processamento paralelo de Migração Reversa no Tempo (RTM) pré-empilhamento

Eng. Desnes Augusto Nunes do Rosário*[1] – engdesnes@gmail.com
Prof. Dr. Samuel Xavier de Souza[1], Profª. Drª. Rosangela Corrêa Maciel[1,3] e Prof. Dr. Jessé Carvalho Costa[2,3]
[1]Universidade Federal do Rio Grande do Norte – UFRN - [2]Universidade Federal do Pará – UFPA – [3]Instituto Nacional de Geofísica do Petróleo – INCT – GP (CNPQ), Brasil

Abstract

This research article presents the analysis of performance and speedup of an initial parallel seismic Reverse Time Migration (RTM) algorithm, employed on the imaging of geological surfaces. Seismic gather and processing of wave data always yield in several variations by factors such as noise, sub-surface inclination, wave energy rounding and severe velocity variations. A well known step in seismic data processing of data is the execution of a migration algorithm, which is used to correct many of these factors. The performance of this algorithm in a pré-stack manner is inversely proportional to the growth of the load of data. For a large load of data, the use of a parallel approach is mandatory to improve the performance of the algorithm. The main goal of this work is to develop a RTM algorithm which is fast and scalable for large data loads.

Introdução

Na indústria de processamento sísmico a migração é um passo computacionalmente custoso dentre diversos outros que são aplicados aos dados. Esta característica ocorre devido à migração ser uma etapa com base em extensas chamadas de entrada/saída de dados 2D e 3D, com uma grande adição de heurísticas matemáticas de alta complexidade algorítmica. Destarte, diversos softwares de migração sísmica requerem um poder de processamento geralmente elevado em consequência da extensa gama de dados envolvida nestes algoritmos.

Um único levantamento sísmico 3D adquirido pela *CGGVeritas* entre 2001 e 2002 na reserva de Tupi foi reprocessado em 2008, e foi observado um aumento significativo na qualidade da imagem do reservatório abaixo do sal. De acordo com seu site, tal reprocessamento foi permitido pela melhoria das técnicas de imageamento e como também pelo surgimento de novas tecnologias do estado da arte.

Por outro lado, ao observarmos este problema sob a perspectiva da engenharia de processadores computacionais, o aumento do desempenho dos núcleos de processamento proposto pela *Lei de Moore*[1] atingiu a barreira física do calor na transmissão de dados em um chip. Neste contexto é correto inferir que a Lei proposta por *Moore* ficou incompleta na década passada, dado o não estabelecimento do aumento de desempenho com o aumento do número de transistores em um chip *single-core*[2] e este foi o escopo necessário para o estopim atrasado da "Era *multi-core*[3]", proposta na década dos anos 70.

Na medida em que esta relativamente nova tendência evolui, centenas de problemas matemáticos das mais diversas áreas, que até o começo da década passada eram dados como não tempo – factíveis, estão sendo repensados e remodelados para novas versões em linguagens paralelas tais como o OpenMP, MPI, CUDA, dentre outras. A aplicação destas linguagens de programação pode aumentar severamente a dificuldade de programação, não obstante, este esforço geralmente pode vir a ser retribuído com uma redução drástica do tempo de execução imediato ou de acordo com o crescimento dos dados, isto é, a carga computacional algorítmica.

O paralelismo e seus paradigmas fazem possível a utilização máxima dos processadores de tecnologia *multi-core,* dado que um software programado serialmente sempre será executado somente por um *core,* por mais esteja sendo executado em um processador *Intel Core i7* ou em até um *cluster*[4].

Nesta conjuntura, este trabalho apresenta a análise de desempenho atual do algoritmo RTM serial e de sua primeira versão paralela, cuja implementação está sendo desenvolvida na linguagem OpenMP. Além disso, serão discutidas as perspectivas futuras da paralelização do código, em especial, sugestões para a melhoria do desempenho e o uso da linguagem CUDA na paralelização do algoritmo.

As imagens das subsuperfícies migradas têm inúmeras aplicações nas áreas de geologia e na geofísica de exploração de petróleo. Entretanto, a qualidade dos resultados deve ter a mesma relevância quanto o tempo requerido para sua obtenção.

[1] Gordon E. Moore, presidente da Intel em 1965: "O número de transistores em um chip terá um aumento de desempenho em 100%, pelo mesmo custo, num período de 18 meses".
[2] Processador formado por um núcleo de processamento
[3] Processador formado por diversos núcleos de processamento. Ex: Dual-core, Core 2 Duo, Quad-core, etc.
[4] Um aglomerado de computadores que utiliza um tipo especial de sistema operacional classificado como sistema distribuído.

Assim, a meta desta pesquisa, atualmente em andamento, é desenvolver um algoritmo paralelo que realize um imageamento de qualidade, em um tempo viável e de forma escalável, como será visto a seguir.

Metodologia/Problema Investigado: Migração Sísmica Reversa no Tempo - RTM

O problema em questão consiste em uma técnica que utiliza os tempos de chegada da energia a superfície, após sua propagação ao subsolo, com o intuito da criação de uma imagem em subsuperfície mais fiel possível a geologia presente no subsolo.

A migração matematicamente consiste numa correção da na posição dos eventos inclinados para as suas supostas posições verdadeiras em subsuperfície, e o colapsamento das difrações em uma seção empilhada (Yilmaz et. al. 2001).

Um levantamento sísmico pode vir a ter centenas ou milhares de tiros sísmicos e até o começo da década passada, a solução mais tempo-factível para o imageamento era a técnica conhecida por migração pós-empilhamento. Esta técnica executa uma rotina somatória dos traços dos tiros sísmicos nas famílias CMP antes de realizar a migração sísmica e isto cria seção empilhada *zero-offset* migrada.

A migração pós-empilhamento é útil quando o modelo de velocidades adquirido é simples, sem severas variações de velocidade. Áreas com uma estrutura complexa e sobrecarregada (grandes variações laterais de velocidade e/ou diapirismo[5]) fazem a suposição hiperbólica associada com os tempos de reflexão não serem mais válidas. Como consequência, a correção do *moveout*[6] hiperbólico e o empilhamento CMP nem sempre levam a uma seção empilhada onde as reflexões das subsuperfícies sejam preservadas fielmente devido a variações de frequência e arredondamentos (Yilmaz et. al. 2001). Desta forma, a migração que proporciona melhor qualidade é a pré-empilhada, que migra cada tiro individualmente ao custo da redução do desempenho do algoritmo.

Os princípios matemáticos da migração obedecem às regras de aditividade e linearidade (inclusive nos casos de *moveouts* não-hiperbólicos), portanto, a seção empilhada migrada pode ser gerada após os cálculos individuais para cada tiro.

Existem algumas formas de abordar um algoritmo de migração sísmica. Os algoritmos de migração sísmica são divididos de três categorias:

- Os baseados em soluções integrais.
- Os baseados em soluções por diferenças finitas.
- Os baseados em soluções implementadas no domínio frequência-número de onda.

[5] Processo de ascensão de diápiro (massa rochosa menos densa que as encaixantes) muitas vezes em estruturas dômicas.

[6] Efeito que ocorre devido ao atraso para a onda chegar ao geofone em função da dispersão radial.

Uma das técnicas que tem mais evoluído junto com o estado da arte da tecnologia é a Migração Reversa no Tempo (RTM – *Reverse Time Migration*).

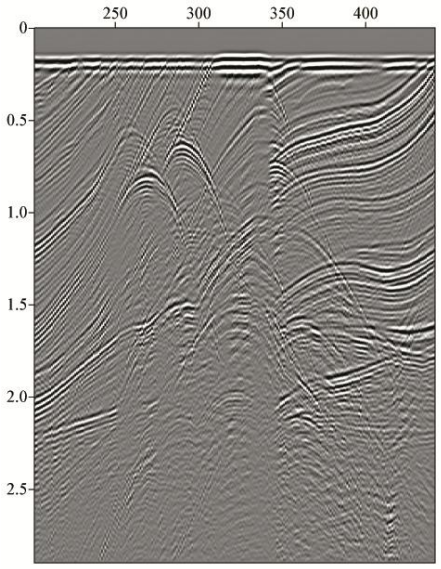

Figura 1 – Seção de offset mínimo - Dados com problemas tais como curvas de difração originadas por estruturas de falhas e domos salinos as quais devem ser corrigidos através da migração sísmica.

Esta técnica entra na categoria de migração por diferenças finitas, pois aplica este esquema de diferenças finitas na resolução da Equação Diferencial Parcial da onda, exibida na Equação (1). (Baysal, 1983)

$$\frac{\partial^2 P}{\partial x^2} + \frac{\partial^2 P}{\partial z^2} - \frac{1}{v^2(x,z)} \frac{\partial^2 P}{\partial t^2} = 0 \quad (1)$$

- x e z são variáveis espaciais.
- t é a variável do tempo.
- v é a velocidade de propagação da onda.
- $P(\mathrm{x}, z, t)$ é o campo de onda de pressão.

Para realizar a migração reversa no tempo de um dado sísmico é preciso fazer a modelagem do seu correspondente modelo de velocidades, desde o tempo $t = 0$ até o tempo $t = t_{max}$, armazenando o resultado em $s(x,z,t)$, para cada ponto do modelo. Em seguida, o dado sísmico é injetado no modelo de velocidades e é propagado desde o tempo $t = t_{max}$ até o tempo $t = 0$ (retropropagação), com o resultado armazenado em $r(x,z,t)$. A condição de imagem aplicada é correlação entre esses campos segundo a Equação (2).
Exemplos do resultado da retropropagação e propagação estão ilustrados nos frames exibidos nas Figuras 3 e 4 respectivamente.

$$I(x,z) = \sum_{t=0}^{tmax} s(x,z,t) r(x,z,t) \quad (2)$$

O RTM realiza a migração dos dados devido às regiões com incidência de uma correlação mais elevada indicar uma troca de meio de velocidade, isto é, uma subsuperfície como ilustra a Figura 5, que exibe o

resultado da correlação de todos os campos propagados e retropropagados, dentre os quais estão os campos das Figuras 2 e 3.

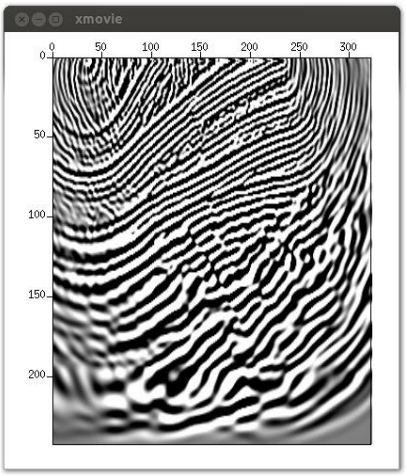

Figura 2 - Um frame de um campo retropropagado

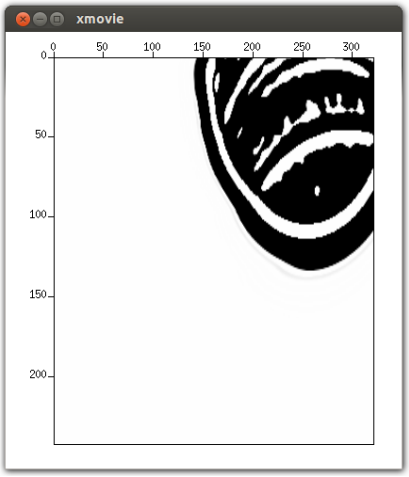

Figura 3 – Um *frame* de um campo propagado

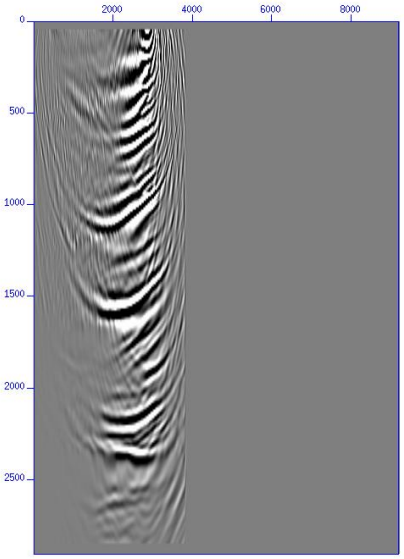

Figura 4 – Imagem migrada de um tiro – resultado da correlação cruzada para um tiro sísmico, fazendo uso de diversos frames como os das Figuras 3 e 4.

Metodologia/Problema Investigado: Speedup, Eficiência e Escalabilidade de Migração Sísmica.

Em programação paralela, uma tarefa de suma importância é o estudo e análise do desempenho de algoritmos. Existem certas métricas cujo intuito é a determinação do melhor algoritmo independente do modelo de hardware disponível. A seguir, serão enunciadas algumas métricas matemáticas de algoritmos concorrentes que foram realizadas nesta etapa inicial da paralelização do RTM.

O *speedup* é um conceito que remete a ideia de quanto mais veloz um algoritmo paralelo é do que sua versão sequencial mais rápida. Matematicamente, o *speedup* pode ser definido pela seguinte relação:

$$S_P = \frac{T_{seq}}{T_{par}(p)} \quad (3)$$

Um *speedup* linear ou ideal é obtido quando ao duplicar-se o número de processadores obtém-se o dobro da velocidade.

Já o conceito de eficiência é definido com uma medida da fração de tempo em que um elemento de processamento é utilmente empregado. Matematicamente definida como a razão do aumento de velocidade para o número de elementos de processamento, isto é, razão do *speedup* pelo número de elementos de processamento, como exibe a equação a seguir:

$$E_f = \frac{S_P}{p} \quad (4)$$

Em um sistema ideal paralelo, o *speedup* com valor igual a p possibilita uma eficiência igual a uma unidade. Na prática, o comportamento ideal não é alcançado porque todo algoritmo sempre terá uma parte serial e também devido a uma CPU não poder entregar 100% do tempo de todos os elementos de processamento para os programas do usuário (GRAMA, GUPTA, KAPYRIS, KUMAR et. al. 2003).

De posse destes conceitos matemáticos de paralelismo, é possível introduzir de uma forma resumida o conceito de escalabilidade, que ocorre quando um algoritmo for adequadamente eficiente e prático de tal forma que seu desempenho aumenta de acordo com que se aumenta a carga computacional (por exemplo, um grande conjunto de dados e modelos de velocidade de entrada tais como os da migração). Em suma, a escalabilidade é a capacidade de um sistema de manter sua eficiência em um valor fixo simultaneamente com o aumento do número de elementos de processamento e o tamanho do problema.

Conceitualmente, a escalabilidade de um sistema paralelo é uma medida da sua capacidade de aumentar seu *speedup* em proporção ao número de elementos de processamento, e a mesma reflete a capacidade de um sistema paralelo de utilizar recursos de processamento

com cada vez maior eficiência (GRAMA, GUPTA, KAPYRIS e KUMAR et. at. 2003).

Tendo estes conceitos como contexto, pode-se introduzir claramente a meta desta pesquisa: A obtenção de um algoritmo escalável pré-empilhado de migração RTM em CUDA, de tal forma que sua eficiência será a melhor possível quando for utilizada uma grande gama de dados. CUDA é uma linguagem que utiliza as unidades de processamento visual embutidas nas placas de vídeo para o aumento de desempenho e do poder de processamento. Ao contrário de programação paralela em CPU's gerida por um Sistema Operacional, a programação feita em GPUs[7] tem uma arquitetura de processamento massivamente paralela, que enfatiza a execução de diversas *threads*[8] simultâneas de uma forma mais lenta, ao invés de enfatizar a execução de uma única *thread* muito rapidamente como nas CPU's, e isto acarreta geralmente em um melhor desempenho.

Outra curiosidade pertinente a este respeito é que o uso da GPU para processamento gráfico nos computadores atuais é mínimo, i. é., a placa de vídeo fica praticamente ociosa a maior parte do tempo a não ser que a mesma esteja executando tarefas que requerem um alto processamento gráfico como vídeos e jogos. A CPU é requisitada constantemente para outras tarefas além de cálculos de programas de usuários como a gerência de processos[9], *threads,* I/O's dentre outros.

Destarte, a programação paralela realizada em GPU, embora mais complexa, geralmente tem um desempenho mais promissor que a realizada em CPU, e esta foi a razão da escolha da linguagem CUDA para abordar este problema no futuro.

Resultados

Este trabalho fez uso do conjunto de dados *Marmousi 2-D*. O *Marmousi data set* que são dados sintéticos gerados pelo *Institut Français du Pétrole* (IFP), simulando uma aquisição marinha 2-D (Versteeg, 1993). A geometria e o modelo de velocidades foram criados para produzir subsuperfícies complexas que requeiram técnicas avançadas de processamento tais como as apresentadas neste artigo.

A equação escalar da onda (Equação 1) precisa de um modelo de velocidades para cada família CMP, como já discutido anteriormente. Estes modelos são processados em partes para somente então serem interpolados espacialmente e este processo resulta na criação de uma matriz de dados de ponto flutuante que representam o valor da velocidade para aquela devida profundidade.

A disposição do arquivo de dados são tiros sísmicos sequenciais, formados por traços estruturados por um cabeçalho padrão de 240 bytes concatenado com N amostras (ponto flutuante) no tempo referente a amostragem da onda captada nos geofones da superfície. O cabeçalho tem informações tais como referencial de tiro, offset no tiro, período de amostragem, dentre diversos outros parâmetros.

Na execução do algoritmo serial foi utilizada uma frequência de 12 Hz para o sinal da fonte e dois modelos de velocidades, um grande com dimensões 2301x751 e um modelo pequeno com dimensões 767x243. O resultado da migração fazendo uso do modelo pequeno de velocidades encontra-se na Figura 5 e os tempos seriais de execução de todo o *Marmousi* fazendo uso dos dois modelos estão ilustrados na Figura 6 (o modelo grande foi estimado).

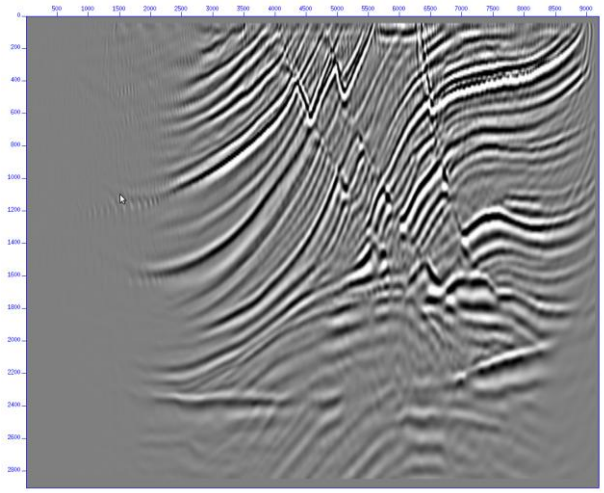

Figura 5 - Resultado da Migração do Marmousi com RTM fazendo uso do pequeno modelo de velocidades

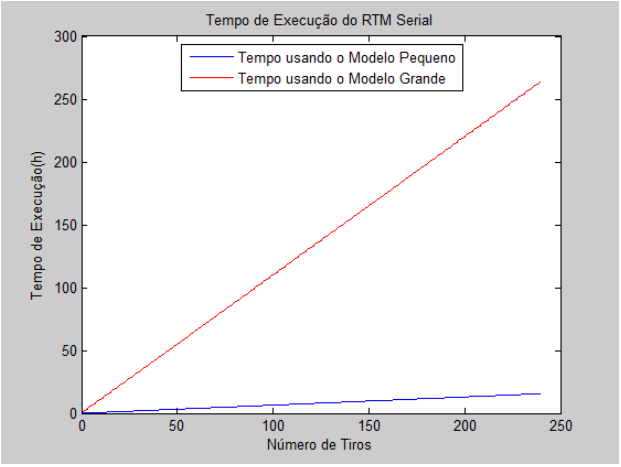

Figura 6 - Desempenho da execução serial usando todo os dois modelos de velocidades

O resultado no quesito de qualidade foi dado como ótimo devido a complexidade do modelo de dados do *Marmousi.* Por outro lado, observa-se o crescimento linear do tempo da execução de acordo com o aumento da carga computacional como exibido na Figura 6. Destarte, é evidenciada a extrema necessidade de uma abordagem paralela para este algoritmo, dado que a

[7] Unidade de Processamento Gráfico (Graphical Processing Unit) embutida nas placas de vídeo da NVIDIA
[8] Um fluxo de execução dentro de um programa em execução
[9] Um programa em execução

execução da migração de todos 240 tiros do *Marmousi* fazendo uso do modelo de velocidades grande, que resultará na máxima qualidade possível desta imagem levar mais que 250 horas, isto é, um período maior que 10 dias de execução ininterrupta.

Após a realização de uma paralelização inicial em uma máquina 24-*core* em OpenMP, com as mesmas especificações de I/O, foi obtido o seguinte desempenho para a execução paralela do primeiro tiro do *Marmousi*:

Tabela 1 – Tabela dos tempos de execução do RTM com diversas threads

Threads	Modelo Pequeno	Modelo Grande
1	0h,3m,53s	1h,6m,1s
2	0h,2m,12s	0h,43m,44s
3	0h,1m,36s	0h,37m,18s
4	0h,1m,18s	0h,32m,19s
5	0h,1m,6s	0h,30m,15s
6	0h,0m,59s	0h,27m,7s
7	0h,0m,53s	0h,24m,47s
8	0h,0m,49s	0h,25m,0s
9	0h,0m,45s	0h,25m,36s
10	0h,0m,42s	0h,23m,42s
11	0h,0m,40s	0h,23m,8s
12	0h,0m,37s	0h,22m,3s
13	0h,0m,36s	0h,20m,20s
14	0h,0m,34s	0h,20m,7s
15	0h,0m,33s	0h,18m,19s
16	0h,0m,31s	0h,17m,43s
17	0h,0m,32s	0h,17m,45s
18	0h,0m,30s	0h,16m,54s
19	0h,0m,29s	0h,16m,54s
20	0h,0m,27s	0h,16m,13s
21	0h,0m,28s	0h,16m,9s
22	0h,0m,27s	0h,15m,38s
23	0h,0m,26s	0h,15m,10s
24	0h,0m,27s	0h,14m,38s

E estes dados resultaram nas Figuras 7, 8 e 9 que representam o tempo de execução, o *speedup* e a eficiência do algoritmo respectivamente.

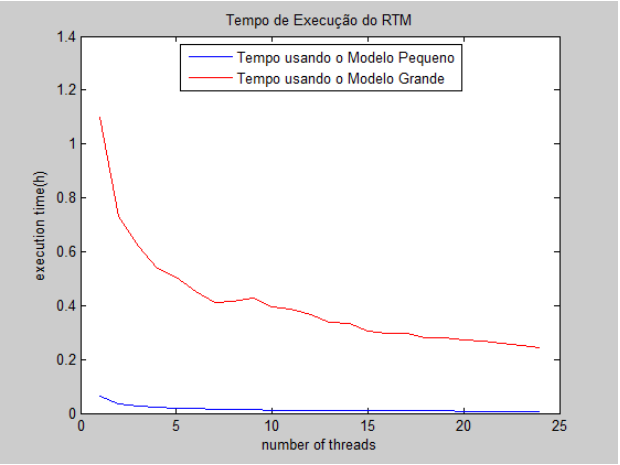

Figura 7 - Tempo de execução do algoritmo paralelo

Pela Figura 7 é possível notar uma redução no tempo de execução por tiro usando ambos os modelos de velocidade. Usando o tempo por tiro, é possível estimar o tempo levado para a execução de todo o modelo de velocidades. Esta estimativa está na tabela abaixo:

Tabela 2 - Tempo de execução de todos os tiros com os dois modelos de velocidades

	Antes da Paralelização	Depois da Paralelização
Pequeno	15 horas e 34 minutos	1 hora e 48 minutos
Grande	264 horas e 6 minutos	58 horas e 35 minutos

Todavia, de acordo com o aumento das threads o tempo de execução converge para uma não melhora do desempenho. A Figura 8 mostra que de acordo com que se aumenta a carga computacional o *speedup* tende a piorar e se distancia do ideal, principalmente com o modelo de velocidades grande. Consequentemente, a eficiência exibida na Figura 9 também reduz drasticamente.

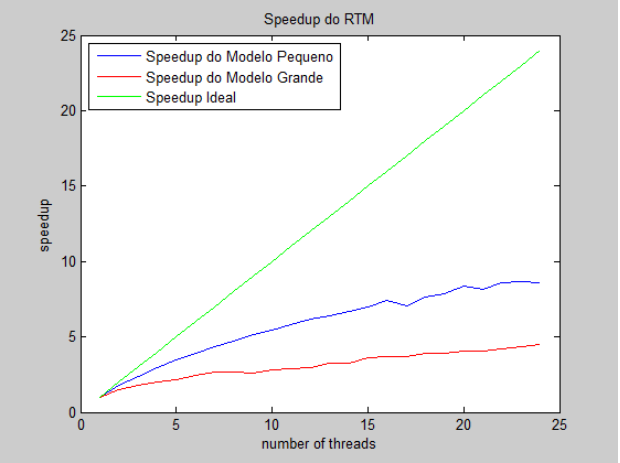

Figura 8 - *Speedup* do Algoritmo Paralelo

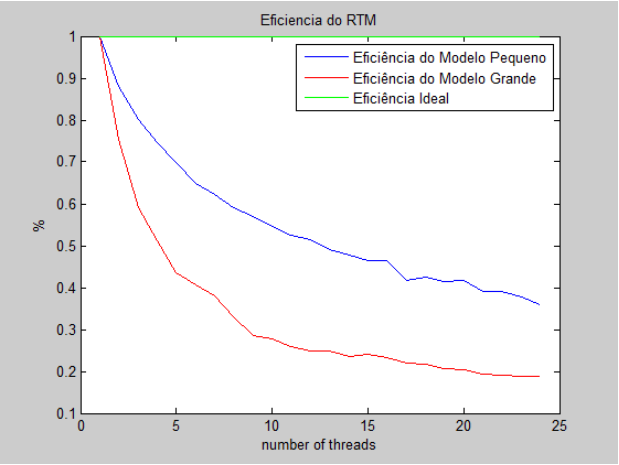

Figura 9 - Eficiência do Algoritmo Paralelo

A razão desta queda desempenho é resultante da estrutura atual do algoritmo, que salva todos os frames retropropagados em um arquivo antes de realizar a correlação cruzada, algoritmicamente definida na Equação (3).

$$(F * D)[N] = \sum_{m=1}^{N} F[m]\, D[N-m] \qquad (3)$$

- F é um *frame* da fonte propagado
- D é um *frame* do dado retropropagado
- N é o nº de *frames* no tiro sísmico

O campo do modelo pequeno retropropagado tem 116 MBytes e o grande tem 1,1 GBytes. Logo, o algoritmo além de ter de calcular todos estes dados, os escreve no disco rígido e os lê reversamente com diversas chamadas de fseek[10] que degradam a desempenho. Por mais que o hardware disponível permita aumentar os elementos de processamento em 24 vezes, o algoritmo realizou aproximadamente um aumento do acesso de dados em escala 10.000 vezes maior. Desta forma, este resultado já era esperado.

Trabalhos futuros

Durante esta paralelização inicial do RTM, foram encontrados diversos trechos no código onde é possível obter melhoras significativas de desempenho. Abaixo, estão descritos alguns pontos onde mudanças na estrutura do algoritmo serão realizadas em breve:

- Acessos desnecessários ao disco rígido
- Acessos desnecessários à memória RAM
- Separação dos processos de retropropagação e propagação
- Execução da Autocorrelação

[10] Função em C que anda com o ponteiro de arquivo para um byte especifico dentro do arquivo

Discussão e Conclusões

Nesta pesquisa, ainda em andamento, concluiu-se que o RTM é uma boa forma de abordar o problema de migração sísmica, dado que foram obtidos resultados de qualidade satisfatória em regiões de estruturas complexas, variações horizontais de velocidade e em áreas de alta incidência de diapirismo. Estes desafios geológicos foram proporcionados pela alta complexidade estrutural do modelo de dados sintéticos *Marmousi.*

Não obstante, foi observado que o desempenho do algoritmo paralelo inicial em OpenMP ainda deixa a desejar, mesmo alcançando uma melhora expressiva no tempo de execução para todo o modelo como mostrado na Tabela 2. Estes dados indicam que uma abordagem paralela eficiente para este problema é mais que necessária para a área da geofísica.

A próxima etapa desta pesquisa é a melhoria da paralelização do código do RTM em OpenMP. A *priori*, é espargido que o resultado da paralelização de cada tiro já resulta em um desempenho altamente promissor. Contudo, é possível crer que a redução da granularidade[11] deste problema resultará em resultados ainda melhores, e com certeza se faz necessário pesquisar até onde esta redução resulta em uma melhora do desempenho. A *pos*teriori, o código em OpenMP será convertido para CUDA permitindo o uso do forte poder de processamento das GPU's.

Após a paralelização dos tiros será analisada a paralelização das heurísticas de retropropagação, propagação e correlação cruzada. Outro problema sempre em consideração será a alocação de memória envolvida nestes processos, já que a memória dos computadores é limitada. Por fim, esta pesquisa será concluída com a prova da possível escalabilidade do algoritmo paralelo de migração sísmica RTM em GPU.

Agradecimentos

Os autores agradecem ao INCT – GP (CNPQ), FAPERN, FINEP e Petrobras. Nunes do Rosário agradece à CAPES pelo apoio financeiro em prol do desenvolvimento científico brasileiro.

Referências

CGGVeritas - *Reverse Time Migration Enhances Tupi Reservoir.* Disponível em http://www.cggveritas.com/default.aspx?cid=2713, acess. no dia 31/07/2012.

Yilmaz Ö., Zhou F., Seismic Data Analysis - Processing, Inversion, and Interpretation of Seismic Data, Tulsa (OK) , USA, vol. 1, 2001: 91-92, 93.

Grana A., Gupta A., Kapyris G., Kumar V., Introduction to Parallel Computing, 2nd edition, 2003: Chapter 5 – Section 5.2 – Desempenho Metrics for Parallel Systems

Baysal, E., D. D. Kosloff, and J. W. C. Sherwood, 1983, Reverse time migration: Geophysics, 48, 1514–1524.

[11] Grau de paralelismo de um problema