FPGA-based Accelerator to Speed-up Seismic Applications

Medeiros, V.W.C.[1]; Rocha, R.C.F. [1]; Ferreira, A.P.A. [1]; Correia, J.C.B.L. [1]; Barbosa, J.P.F. [1]; Silva-Filho, A.G. [1]; Lima, M.E. [1]; Rodrigo Gandra[2] and Ricardo Bragança[2]
Centro de Informática (CIn) - Universidade Federal de Pernambuco - Recife - Brasil[1]
CENPES/Petrobrás[2]
{vwcm, rcfr, apaf, jcblc, jpfb, agsf, mel}@cin.ufpe.br[1]
{rodrigo.gandra, rbraganca}@petrobras.com.br[2]

Abstract

Hardware accelerators such as GPGPUs and FPGAs have been used as an alternative to the conventional CPU in scientific computing applications and have shown significant performance improvements. In this context, this work presents an FPGA-based solution that explores efficiently the reuse of data and parallelization in both space and time domains for the first computational stage of the RTM (Reverse Time Migration) algorithm, the seismic modeling. We also implemented the same algorithm for CPU architectures and GPGPU and our results demonstrate that the FPGA-based approach can be a viable solution to improve performance. Experimental results show a speedup of 1.668 times compared with GPGPU and 25.79 times compared to CPU. Results were evaluated with the Marmousi velocity model, considering the same parameters in all approaches.

1. Introdução

Atualmente, os modelos de simulação para reconhecimento de pontos estratégicos de perfuração e extração de petróleo são extremamente complexos e necessitam de alto poder de processamento, pois podem levar meses para simulação completa de uma certa região. Ao mesmo tempo, a competitividade com as descobertas de petróleo no Brasil na camada de pré-sal, estimam-se volumes de cerca de 10 bilhões de barris de petróleo [1], favorecem a implementação de novas estratégias para acelerar a definição de pontos propícios de perfuração.

Recentemente, aceleradores tais como GPGPU e FPGAs têm surgido como fortes candidatos a computação em *streaming* [3][4][5][7] para aumentar o desempenho em uma plataforma de hardware específica. Com isso, algoritmos de computação intensiva aplicados a modelagem de computação científica tais como bioinformática, processamento de imagem, geofísica e aplicações sísmicas [2][9] podem ser mapeados diretamente em FPGAs [6][7] ou GPGPUs [4][3] visando explorar o poder de processamento massivo.

Particularmente, as plataformas reconfiguráveis têm sido vastamente utilizadas em aplicações de processamento digital de sinal em aritmética de ponto fixo, alcançando ganhos em desempenho quando comparados com as CPUs de propósito geral [11][7] e até GPGPUs [3][4].

De fato, avaliar qual é a plataforma ótima para uma dada aplicação tem se tornado uma tarefa difícil. Por exemplo, aplicações tais como imageamento sísmico na indústria de petróleo, foco principal deste trabalho, envolvem uma grande quantidade de dados baseados em operações de ponto flutuante nos campos de petróleo, visando localizar reservatórios de óleo e gás no subsolo, com base em fontes de ondas sísmicas, geofones e um sistema de armazenamento de dados. Este processo é feito através da aquisição de tempo e velocidade de viagem das ondas sísmicas na qual podem ser reconstruídas por aplicar métodos tais como Kirchhoff [14] e RTM [2][8], resultando em dados que são usados para verificar a disposição das rochas na terra.

As contribuições deste artigo estão direcionadas a: (i) análise e desenvolvimento de uma arquitetura para acelerar o método RTM de uma modelagem sísmica 2D para 3 diferentes abordagens: CPU, GPGPU e FPGA; (ii) Implementação em hardware na plataforma Gidel; (iii) Implementação parametrizável do problema em hardware.

1.1. Algoritmo RTM

Exploração sísmica de óleo pode ser dividida em 3 áreas principais: aquisição, processamento e interpretação. A aquisição é responsável pela obtenção

dos traces sísmicos, através da injeção de uma fonte de excitação (solo ou água) e posterior captura dos sinais refletidos através de receptores (geofones ou hidrofones) como pode ser mostrado na Figura 1.

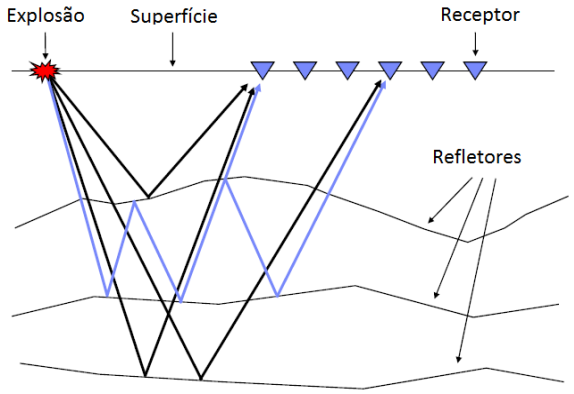


Figura 1 - Visão geral da reflexão sísmica

O processamento inclui vários algoritmos tais como Kirchhoff e RTM capazes de transformar uma informação confusa obtida nos conjuntos de traços sísmicos (sismograma) em algo compreensível por especialistas. No estágio de interpretação, com base na informação de vários níveis geológicos extraídos de uma seção sísmica processada, as imagens geradas resultantes são finalmente analisadas por geólogos e geofísicos. O desenvolvimento deste trabalho está direcionado no passo de processamento, particularmente na modelagem sísmica.

Alguns métodos de modelagem e migração sísmica têm sido aplicados em diferentes áreas geológicas. O método de Kirchhoff tem sido o mais usado na indústria de petróleo, devido ao seu baixo custo computacional comparado a outros métodos existentes. No entanto, este método não é adequado para áreas que têm estruturas muito complexas, com variações laterais em velocidade, ou altos mergulhos das camadas. O método RTM permite resolver diretamente a equação da onda acústica/elástica com resultados bem mais precisos. No entanto, este método tem um alto custo computacional. Neste sentido, torna-se necessário estratégias para aumentar o desempenho de tais aplicações.

2. Trabalhos Relacionados

Com o objetivo de melhorar o desempenho do sistema sobre aplicações de computação científica, plataformas paralelas diferentes têm sido avaliadas para diferentes algoritmos. Têm sido significante os debates sobre qual plataforma produz uma solução mais eficiente. No entanto, apenas através de uma otimização cuidadosa de cada plataforma, associado a expertise de especialistas no hardware envolvido, pode-se extrair uma solução otimizada para o problema.

Para explorações na indústria de petróleo, pesquisas têm sido intensas uma vez que processamento de dados sísmicos usam algoritmos muito complexos com operações em ponto flutuante, indicados por especialistas geofísicos para reconstruir o subsolo da terra para descobrir onde pode ser encontrado petróleo.

Com isso, todos os anos, várias soluções estratégicas de alto desempenho, usando diferentes arquiteturas têm sido propostas. As otimizações são as mais diversas possíveis tais como acesso a memória, freqüência de operação, elementos de processamento, redução de precisão, dentre outros. Todas essas características, associadas com a mais avançada tecnologia, não teria sucesso algum se a criatividade de cada projetista em prover uma solução diferente para computação em *streaming* não fosse avaliada.

Chuan He et al. [6] propôs uma plataforma reconfigurável para processamento de dados sísmicos baseados em FPGA para acelerar o algoritmo PSTM (*Pre-Stack Kirchhoff Time Migration*). Este algoritmo é caro para aplicações 3D porque ele é computacionalmente intensivo e requer uma grande quantidade de dados de entrada. A parte do kernel do algoritmo que consome quase 90% do tempo de CPU foi ajustada para uma plataforma baseada em um FPGA da Xilinx Virtex II Pro Series [15]. Resultados de simulação operando a uma frequência de 50MHz mostram que a solução proposta é 15.6 mais rápida que uma workstation P4 2.4GHz.

O mesmo algoritmo PSTM foi re-implementado em uma GPGPU GeForce8800GT NVidia [16] por Shi et al. [14]. O protótipo foi avaliado e resultados mostraram que eles são 7.2 vezes mais rápidos que uma CPU Intel P4 3.0 GHz. É importante mencionar que existem GPUs mais novas que as 8800s, uma delas será avaliada na proposta deste trabalho.

Clapp, R. et al. [2] fazem uma discussão do algoritmo RTM para aceleradores GPU e FPGA. O trabalho discute algumas decisões de otimização a serem feitas em cada plataforma levantando os aspectos positivos e negativos em cada abordagem.

O trabalho de Thomas, D. et al. em [3], avalia GSample/s e MSample/Joule para alguns algoritmos distintos de geração de números aleatórios (Uniforme, Gaussiano, Exponencial) em diferentes arquiteturas (multi-core CPUs, GPUs, FPGAs e MPPAs). Resultados mostram que FPGA provê uma ordem de magnitude a mais comparado as outras plataformas e cerca de 250 vezes mais rápido que a CPU.

Este trabalho explora o uso de FPGA para realizar a aceleração do problema de modelagem sísmica 2D. A infra-estrutura usada será explicada na próxima sessão.

3. Arquitetura do Sistema

Para a implementação em FPGA utilizamos a plataforma da Gidel PROCe-III [12] como ambiente experimental a qual é composta de um FPGA Altera Stratix III 260E [13] conectado a um host Intel Core 2 Quad através de um barramento PCIe x4 e pode ser ilustrado através da arquitetura da Figura 2.

O *host* é composto por: o driver da aplicação, uma API para acessar os recursos do co-processador e um *device driver*. O co-processador é composto por: o core de uma ponte PCIe, implementado em um FPGA adicional; o FPGA principal, que suporta *wrappers* da Gidel; um core Multiport que age como um nível de abstração entre módulo de memória e core do usuário, provendo um caminho simples de leitura e escrita; um core de processamento em *streaming* (Arch2D), que implementa o algoritmo de modelagem sísmica RTM e memória DDR2.

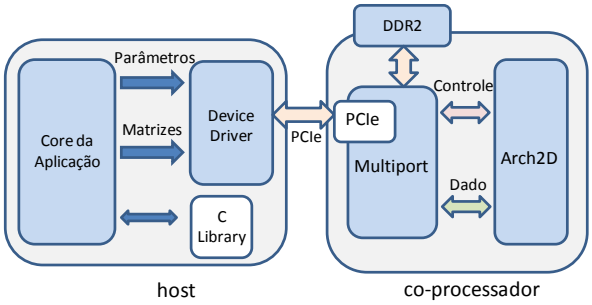


Figura 2 - Arquitetura do sistema

A placa FPGA é composta de elementos de processamento (PEs) essenciais para maximizar o paralelismo no algoritmo RTM através do método de diferenças finitas, visando uma arquitetura de processamento em *stream* do algoritmo RTM. Esta estrutura compreende um módulo Arch2D responsável por interconectar todos os módulos do FPGA e controlar o fluxo de dados para os núcleos de processamento.

A quantidade de PEs é um fator que necessita ser avaliado, pois depende de vários aspectos tais como: a área que o FPGA pode suportar, o fornecimento de dados para processamento em *streaming*, frequência de operação, dentre outros aspectos. Neste sentido, e considerando as limitações da placa disponível usamos N=4 (ou seja, 4 PEs) e uma largura de banda de 128 bits para acesso a memória DDR2 como indicado na Figura 3.

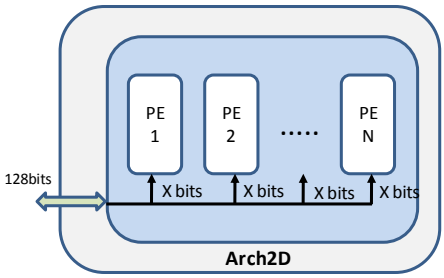


Figura 3 - Visão geral do Arch2D

Um tipo de otimização que ajudaria bastante na relação largura de banda/quantidade de PEs seria no tamanho da palavra para acesso a memória. Acesso a memória para computação intensiva ainda é um gargalo que pode perdurar por um bom tempo. Neste sentido, ajustar o formato da palavra pode ser uma boa estratégia para otimizar a quantidade de palavras buscadas na memória e conseqüentemente uma melhoria no desempenho da aplicação. Este trabalho não aborda esta estratégia, no entanto, nosso grupo de trabalho vem trabalhando paralelamente com este tipo de otimização.

A aplicação em software está direcionada à execução do algoritmo de modelagem RTM. O módulo principal desta aplicação é composta por algumas funções básicas. A primeira dessas funções é dedicada a organizar os dados, do disco do *host*, para as memórias conectadas ao FPGA (DDR2). A base de dados do algoritmo RTM é basicamente composta de 3 matrizes: a matriz de velocidade (VEL); a matriz de campo de pressão atual (APF); e a matriz de campo de pressão anterior (PPF).

A matriz VEL armazena o modelo, proposto por um geólogo, que representa diferentes velocidades de propagação de onda nos níveis da terra. Como o modelo é um algoritmo de diferenças finitas temporal de segunda ordem, duas matrizes são envolvidas no processamento, APF (que armazena o valor do campo de pressão do passo atual) e PPF (que armazena o valor do campo de pressão do passo anterior).

Essas três matrizes são confinadas em blocos de memória de tal forma que o acesso rápido aos dados em forma de streaming seja possível. Esta organização de dados é explicada em mais detalhes nas próximas seções.

Outra função do core da aplicação é configurar alguns parâmetros tais como tamanhos de matrizes, geração de dados de pulsos sísmicos e número de *time steps*. Por último, o core da aplicação também tem a função de visualizar os dados processados.

Usando a IDE PROCWizard da Gidel foi desenvolvido uma interface simples para integrar componentes hardware e software do projeto. O PROCWizard trabalha junto com a placa da Gidel e é capaz de gerar código de hardware e software, em HDL e C++ respectivamente, e um canal de comunicação através de um barramento PCIe. Esta IDE também integra automaticamente todos os IP-Cores da Gidel para o projeto, assim como o PROCMultiport para gerar uma abstração de acesso a dados da DDR2, bem como um *device driver* usado pelo core da aplicação para acessar os recursos da placa e debug do ambiente em tempo de execução.

Esta placa de FPGA possui 3 bancos de memória disponíveis: uma DDR2 512 MB SDRAM e 2 DDR2 SODIMM com 4GB cada uma. Todas as memórias tem uma interface de 64 bits e acessíveis via o core PROCMultiport da Gidel. Algumas otimizações foram implementadas de fato neste trabalho e serão detalhadas posteriormente.

3.1. Implementação do *Streaming Processing*

O core de processamento em streaming é composto de elementos de processamento (PEs) e estruturas de suporte responsáveis por prover dados eficientemente para eles. Esses elementos de processamento implementam a equação de diferenças finitas. Esta equação tem um alto grau de paralelismo devido a interdependência no cálculo de cada um dos elementos. Esta característica permite uma grande melhoria de desempenho através da instância de múltiplos elementos de processamento no core de processamento em *streaming*. Como indicado anteriormente, foram instanciados 4 PEs na arquitetura proposta. Isso significa que 4 pontos da matriz podem ser processados ao mesmo tempo. Além dos PEs, a estrutura interna do core de processamento possui módulos responsáveis pela inserção do pulso sísmico ao algoritmo e a unidade de controle.

A organização dos dados em memória foi feita de tal forma a reduzir a latência de acesso. A idéia principal é organizar os dados de uma forma que permita o acesso sequencial, fazendo com que o controlador de memória trabalhe em modo rajada (*burst mode*), e consequentemente, diminuir a latência de memória.

A Figura 4 mostra a distribuição dos dados dentro da memória DDR2 da placa. Inicialmente, a memória armazena o vetor de pulso sísmico e em seguida as matrizes APF, PPF e VEL. O vetor de pulso sísmico é uma representação de uma fonte de excitação aplicada ao modelo sísmico e é disposto seqüencialmente na memória. As matrizes são divididas em *slices* e cada *slice* corresponde a 4 colunas na matriz original. O que significa que o primeiro *slice* tem todas as linhas das primeiras 4 colunas. Pode-se considerar que a memória é uma sequência de palavras de 128 bits. Como o PROCMultiport provê uma interface de 128 bits de acesso à memória DDR2, cada leitura de uma palavra corresponde a 4 pontos de ponto flutuante com precisão simples (32 bits), que representa uma única linha do *slice*. Todas as matrizes têm o mesmo tamanho e o vetor de pulso sísmico tem um tamanho igual ao número de *time steps* que estão sendo processados.

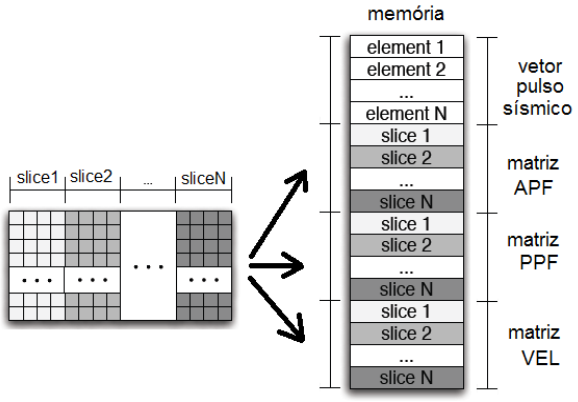


Figura 4 - Organização de dados na memória

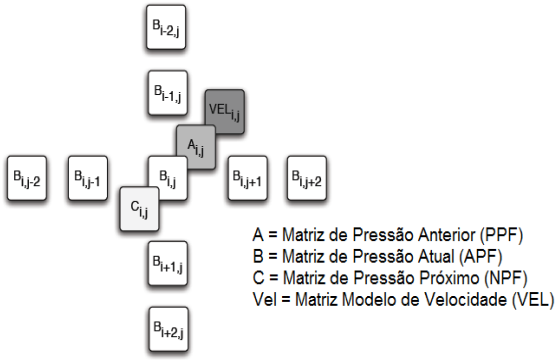
Devido à largura de banda de memória e as restrições de recursos internos do FPGA, 4 PEs foram instanciados no *core* de processamento. A cada ciclo de clock, o PE lê suas portas de entrada (PPF, APF, VEL) indicadas através da equação indicada na Figura 5. Em seguida, é gerado o próximo valor do campo de pressão (NPF). Com isso, a cada passo de processamento, 4 novos pontos da matriz são calculados em paralelo.

As estruturas de FIFOs e *shift registers* têm duas funções básicas no sistema. Primeiro é prover dados corretamente e de forma eficiente para os PEs e o segundo é implementar o reuso de dados.

A idéia principal é processar 4 elementos adjacentes ao mesmo tempo. Cada um deles em um PE específico. Usando esta estratégia, o estêncil torna-se algo como ilustrado na Figura 5. Com isso, uma estrutura de registradores que alimentam estes 4 elementos de processamento é criada.

Uma grande vantagem da arquitetura proposta é o reuso de elementos adjacentes. Se não considerarmos o reuso, cada elemento de processamento tem 9 entradas da matriz APF, 1 entrada da matriz PPF e 1 entrada da matriz VEL. Com isso, para alimentar os 4 elementos de processamento 44 pontos da matriz deveriam ser

lidos. No entanto, como alguns dados são comuns aos elementos de processamento adjacentes e devido ao armazenamento interno destes dados através das estruturas de FIFOs e shift registers, somente 12 novos elementos da matriz devem ser carregados para cada passo de processamento. Isto significa uma redução de aproximadamente 72% no acesso a memória externa.



$$C_{i,j} = 2 * B_{i,j} - A_{i,j} - \{Vel^2_{i,j} * fat * [16 * (B_{i,j+1} + B_{i,j-1} + B_{i+1,j} + B_{i-1,j}) - 1 * (B_{i,j+2} + B_{i,j-2} + B_{i+2,j} + B_{i-2,j}) - 60 * B_{i,j}]\}$$

Figura 5 - Equação de propagação da onda e estêncil

Para gerenciar todo o fluxo de dados para dentro do *core*, uma unidade de controle foi desenvolvida. Além de gerenciar o tráfego de leitura dos parâmetros da aplicação do *host* e as matrizes de entrada através do core PROCMultiport, o módulo de controle é responsável por escrever as saídas dos PEs de volta na memória.

3.2. Otimizações

A implementação inicial não é a mais eficiente com relação ao uso da largura de banda em memória. No cenário inicial foi considerada a utilização de apenas um único banco de memória. Contudo, a placa da Gidel possui 3 bancos de memória sendo 2 bancos de 4 GB e mais um com 512 MB. Na primeira otimização passamos a utilizar todos os bancos de memória disponíveis. Quando todos os bancos de memória são usados, a largura de banda de memória aumenta em 3 vezes. Nesta disposição, a grande vantagem é que as matrizes PPF, APF e VEL podem ser acessadas concorrentemente, diferente do primeiro cenário que era acessado uma palavra por vez no mesmo banco de memória.

Uma vez que o acesso a memória melhorou em 3 vezes, em teoria, é possível aumentar a freqüência de clock também em 3 vezes, melhorando dessa forma o desempenho geral do sistema. Contudo, os bancos de memória da placa Gidel não são totalmente semelhantes, eles possuem capacidades e frequências de operação diferentes. Por esta razão, esta otimização foi capaz de aumentar a frequência de clock de 50 MHz para 120 MHz gerando um ganho de desempenho real de 2,4 vezes ao invés da estimativa teórica de 3 vezes.

Uma outra otimização considerada foi de implementar uma abordagem com *pipeline* temporal. As limitações desta abordagem são impostas principalmente pela quantidade de memória interna ao FPGA e o número de blocos lógicos. A vantagem de processar vários passos temporais paralelamente é que está abordagem não aumenta a necessidade de largura de banda no acesso aos dados, pois os dados lidos da memória RAM para o processamento do primeiro passo temporal são armazenados nas memórias internas do FPGA e passados para as unidades que processam os passos temporais seguintes. Por outro lado, o número de elementos de memória internos ao FPGA são restritos impondo uma limitação ao número de passos temporais que podem ser calculados simultaneamente. Para o FPGA utilizado na nossa implementação não é possível alocar mais do que 4 passos temporais. A Figura 6 mostra a estratégia de *pipeline* temporal implementada nesta otimização.

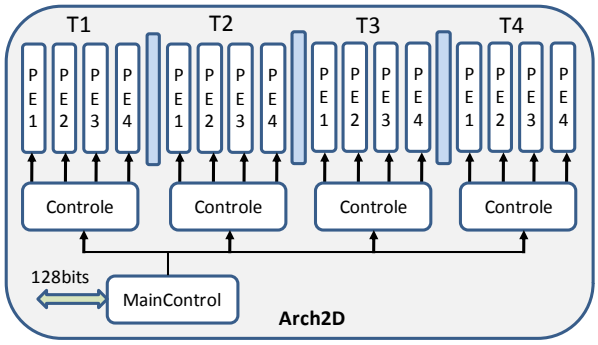


Figura 6 - Otimização de *pipeline* temporal (profundidade=4)

Cada passo temporal (T1, T2, T3 e T4) indicado na Figura 6, possui um conjunto de 4 PEs. As saídas dos PEs dos passos T1, T2 e T3 alimentam FIFOs que são responsáveis por armazenar os dados que serão utilizados no processamento dos passos seguintes. Por fim, a saída dos PEs do passo T4 é armazenada na memória RAM da placa. Módulos de controle são responsáveis pelo gerenciamento do fluxo destes dados. Sinais de controle se comunicam com um módulo de controle principal (*MainControl*) no Arch2D, que é responsável pela comunicação com o

PROCMultiport para leitura e escrita na memória da placa.

O PROCMultiport age como um nível de abstração entre os módulos de memória e o *core* do usuário. Cada porta de comunicação possui 128 bits de largura de banda. Na nossa implementação são utilizados 3 portas (APF, PPF e VEL). Neste sentido, outros mecanismos de otimização que permitam acessar mais dados com a mesma largura de acesso ajudaria a aumentar o desempenho do sistema.

Técnicas tais como compressão de dados e redução de precisão vêm sendo estudadas em paralelo pelo nosso grupo de pesquisa.

4. Resultados

No intuito de avaliar o desempenho do sistema desenvolvido, nós implementamos o mesmo algoritmo de modelagem RTM em uma GPGPU Tesla T10 [17] e em um processador AMD Athlon 64 X2. Apesar de termos GPGPUs mais novas disponíveis no mercado, a Tesla T10 ainda apresenta um excelente desempenho. Além disso, não estamos utilizando também o modelo mais recente de FPGA. A implementação em CPU não utiliza o set de instruções SSE. Em nossos experimentos foi utilizado o modelo de velocidades Marmousi [18].

Os parâmetros mais importantes na execução do algoritmo são os tamanhos das matrizes e o número de *time steps* que serão processados. Originalmente, o modelo de velocidades tem 751 linhas e 2301 colunas, e pode ser visualizado na Figura 7.

A implementação do algoritmo para a plataforma GPGPU foi realizada na linguagem OpenCL e compilada através do compilador fornecido pela NVIDIA.

No FPGA os dois cenários com e sem otimizações foram avaliados.

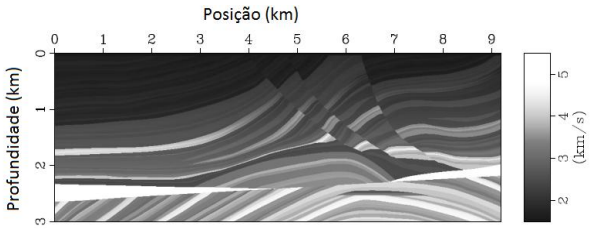


Figura 7 - Modelo de velocidade Marmousi

O desempenho medido em Gsamples/segundo para as implementações em CPU, GPGPU, FPGA não otimizado e otimizado, para todos os experimentos, é apresentado na Figura 8. Para os experimentos, subconjuntos do modelo Marmousi foram utilizados fixando o número de linhas e variando o número de colunas na seguinte sequência: 800, 1000, 1200, 1400, 1600, 1800, 2000, 2200 e 2300. É possível perceber na Figura 8 que a variação do tamanho do experimento pouco influencia no desempenho das plataformas exploradas.

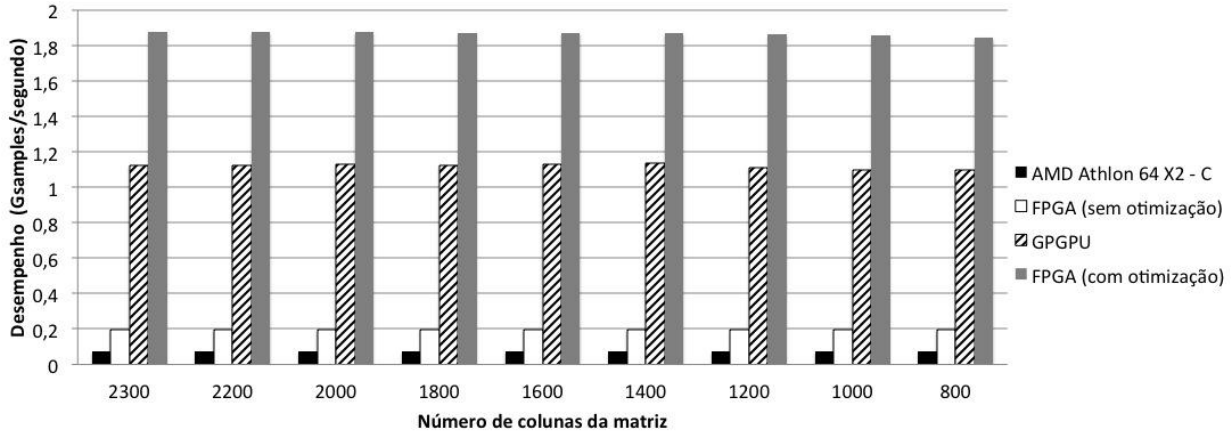


Figura 8 - Tempo de Processamento de FPGA, GPGPU e CPU sem otimizações

A Tabela 1 mostra a relação de *speed-up* entre as arquiteturas. Os valores nesta tabela são um valor médio de todos os experimentos. A implementação otimizada em FPGA é aproximadamente 25,79 vezes mais rápido que a CPU e 1,668 vezes maior que a GPGPU. Também é possível perceber que a flexibilidade do FPGA aliada a uma boa estratégia de otimização permitiu uma melhoria de desempenho de quase 10 vezes em relação a solução não otimizada. Novas otimizações que estão sendo desenvolvidas pelo

nosso grupo de pesquisa como redução de precisão e compactação de dados podem aumentar ainda mais o desempenho.

Tabela 1 - Relação de *Speed-up* entre CPU, GPGPU e FPGA com e sem otimizações

	CPU	FPGA	GPGPU	FPGA otim.
CPU	-	0,366	0,065	0,039
FPGA	2,731	-	0,177	0,106
GPGPU	15,465	5,663	-	0,600
FPGA otim.	25,790	9,444	1,668	-

5. Conclusões e Trabalhos Futuros

Neste trabalho, nós apresentamos uma arquitetura para melhorar o desempenho de uma aplicação sísmica baseada em FPGA referente a etapa de modelagem do algoritmo sísmico RTM. Uma versão baseada em GPGPU e CPU do algoritmo sísmico também foi desenvolvido para avaliar a performance da abordagem baseada em FPGA.

Em uma primeiro cenário, foi observado que o FPGA é mais rápido que a CPU mas mais lento quando comparado com GPGPU. Nesta primeira abordagem nenhuma otimização foi considerada para a abordagem FPGA. No entanto, algumas possibilidades de melhorias para FPGAs tais como o uso da largura de banda de memória e paralelização no domínio do tempo foram apresentados e implementados. Apesar do fato de que a implementação de paralelização no domínio espacial na GPGPU é simples quando se usa OpenCL, otimizações tais como paralelização no domínio do tempo e mudança de precisão de ponto flutuante são muito custosas ou até impossíveis de implementar. Com isso, uma implementação em FPGA oferece muito mais potencial para otimizações mais sofisticadas.

Resultados para otimização em FPGA é, em média, 1,668 vezes mais rápido do que GPGPU e 25,79 vezes mais rápido do que CPUs. Considerando que nem todas as otimizações do FPGA foram consideradas, ela pode ser considerada uma abordagem bastante promissora.

Trabalhos em andamento têm sido desenvolvido pelo nosso grupo de pesquisa visando integrar novas otimizações tais como compressão de dados e redução de precisão no formato do ponto flutuante visando melhorar ainda mais o desempenho da aplicação sem comprometer a qualidade dos resultados.

Uma avaliação de consumo de energia para a modelagem sísmica também está sendo desenvolvida. Como o FPGA trabalha a frequências cerca de 10 vezes menores que GPGPUs e CPUs, o reduzido consumo de energia pode ser outra grande vantagem na solução baseada em FPGA.

6. Agradecimentos

Os autores gostariam de agradecer ao Centro de Pesquisa da Petrobrás (CENPES) pelo suporte técnico conceitual sobre modelagem sísmica. Adicionalmente gostaríamos de agradecer à coordenação da Rede RPCMod e FACEPE pelo suporte financeiro parcial ao projeto.

7. Referências

[1] Estatísticas da Petrobrás referentes ao Pré-Sal. Disponível em: http://www.petrobras.com.br/presal/. Acessado em Junho de 2011.

[2] R. G. Clapp, H. Fu, and O. Lindtjorn. "Selecting the right hardware for Reverse Time Migration". In: The Leading Edge 29: 48-58, January 2010.

[3] Thomas, D.B.; Howes, L.; Luk, W. "A Comparison of CPUs, GPUs, FPGAs, and Massively Paralell Processor Arrays for Random Number Generation". FPGA 2009, pp.63-72.

[4] S. Che, J. Li, J.W. Sheaffer, K. Skadron, and J. Lach. "Accelerating Compute Intensive Applications with GPUs and FPGAs". In: Proc. Symp. Application Specific Processors, pp. 101-107, 2008.

[5] M. B. Gokhale and P. S. Graham. "Reconfigurable computing: Accelerating computation with field-programmable gate arrays". Springer-Verlag, ISBN: 0387261052. New York, 2005.

[6] C. He, G. Qin, M. Lu, W. Zhao. "Optimized high-order finite difference wave equations modeling on reconfigurable computing platform". In: Microprocessors & Microsystems, v. 31 , pp.103-115, March 2007.

[7] S. Brown, "Performance Comparison of finite-difference modeling on Cell, FPGA and multi-core computers", *SEG*, SEG Publisher, San Antonio, 2007, pp. 2110-2114.

[8] W. W. Symes, \Reverse time migration with optimal checkpointing," Geo-physics, vol. 72, no. 5, pp. SM213{SM221, 2007. [Online]. Available: http://link.aip.org/link/?GPY/72/SM213/1

[9] C. Petrie, C. Cump, M. Devlin, and K. Regester, "High performance embedded computing using Field Programmable Gate Arrays", In: Proceedings of the 8th Annual Workshop on High-Performance Embedded Computing, 2004, pp. 124-150.

[10] H. Fu, W. Osborne, R. G. Clapp, O. Mencer and W. Luck. “Accelerating Seismic Computations Using Customized Number Representations of FPGAs” EURASIP Journal on Embedded Systems, vol. 2009, Article ID 382983, doi: 10;.1155/2009/382983, pp. 1–13, 2009.

[11] C. Chang, J. Wawrzynek, and R. Brodersen, "Bee2: a high-end reconfigurable computing system," Design Test of Computers, IEEE, vol. 22, no. 2, pp. 114-125, mar. 2005.

[12] Plataformas da GiDEL: PROCe III. Disponível em: http://www.gidel.com/PROCe%20III.htm. Acessado em Junho de 2011.

[13] FPGA Stratix III da Altera. Disponível em: http://www.altera.com/products/devices/stratix-fpgas/stratix-iii/overview/st3-overview.html. Acessado em Junho de 2011.

[14] O. Yilmaz, Seismic Data Analysis. Tulsa, OK: Society of Exploration Geophysicists, 2001. [Online]. Available: http://link.aip.org/link/doi/10.1190/1.9781560801580.

[15] FPGAs da Xilinx. Disponível em: http://www.xilinx.com. Acessado em: 22/06/2011.

[16] GPGPU da NVidia. Disponível em: http://www.nvidia.com. Acessado em: 22/06/2011.

[17] GPGPU Tesla T10. Disponível em: http://www.nvidia.cn/docs/IO/56483/Tesla_C1060_boardSpec_v03.pdf. Acessado em: 14/07/2011.

[18] Trevor Irons. Marmousi Model. <http://www.ahay.org/RSF/book/data/marmousi/paper.pdf>. Consultado em: 18 de Julho de 2010.