MODELO COMPUTACIONAL HETEROGÊNEO PARA INVERSÃO DA FORMA DE ONDA COMPLETA 2D EM SISTEMAS MULTICORE E MULTIGPU

Victor Koehne[1]; Murilo Boratto[2]; Oscar Mojica[3]
[1] Bolsista PD&I 2 – EMBRAPII; victor.ramalho@fbter.org.br
[2] Doutor em Ciência da Computação; Centro de Supercomputação SENAI CIMATEC; Salvador-BA; murilo.boratto@fbter.org.br
[3] Doutor em Geofísica; Centro de Supercomputação SENAI CIMATEC; Salvador-BA; oscar.ladino@fieb.org.br

RESUMO

A Inversão da Forma de Onda Completa (FWI - do inglês *full-waveform inversion*) é um dos processos mais utilizados atualmente para determinar parâmetros da subsuperfície com alta resolução, que só é factível através de implementações paralelas. Usualmente, essas implementações fazem uso de paralelismo multicore ou multi-GPU, mas não ambos. Neste trabalho é proposto um modelo híbrido que faça uso simultâneo de recursos multicore e multi-GPU. São apresentadas as implementações multicore e multi-GPU com base na literatura, e introduzida a implementação híbrida. É feita uma prova de conceito num modelo de velocidades bem estabelecido, na qual os resultados do algoritmo híbrido são comparados aos do algoritmo multi-GPU.

PALAVRAS-CHAVE: multicore, multi-GPU, heterogêneo, FWI.

1. INTRODUÇÃO

Um modelo de alta resolução da subsuperfície é crucial para evitar erros na perfuração de poços de óleo e gás, bem como aumentar a produção de áreas já exploradas, especialmente para o caso de reservatórios situados em ambientes de geologia complexa, como o pré-sal brasileiro ou o Golfo do México. A inversão da forma de onda completa (FWI) vem se tornado uma das principais ferramentas para estimar modelos de velocidades sísmica de alta resolução. A FWI é um problema de alta intensidade computacional, somente praticável por meio de implementações em computação paralela.

Implementações usuais de FWI 2D num cluster de CPUs consiste em processar o máximo de tiros possível em paralelo, enquanto a extrapolação dos campos de onda é serializada em cada CPU, e por vezes otimizada por diretivas OpenMP[1]. Por outro lado, a implementação mais usual em GPU consiste em calcular cada ponto do campo de onda em paralelo, um tiro de cada vez[2]. Em nossa abordagem, essas duas ideias são fundidas e estendidas a fim de alavancar o máximo de recursos computacionais dos núcleos de CPU e todos as placas GPU disponíveis, na implementação da FWI, de forma a alcançar uma alta eficiência de maneira automática através de uma distribuição apropriada de *workloads* entre todos os recursos computacionais.

2. METODOLOGIA

Do ponto de vista computacional, a FWI pode ser vista como um processo que executa uma migração reversa no tempo (RTM - do inglês *reverse time migration*) por iteração. A implementação da RTM aqui utilizada constitui-se de três processos de modelagem de propagação de ondas sísmicas no modelo de velocidades 2D (simplesmente denominados “modelagem” a partir deste ponto). Um processo extra de modelagem é necessário para o cálculo do passo do algoritmo do tipo gradiente utilizado na FWI[2]. Desta forma, o maior custo computacional de uma iteração da FWI corresponde a quatro modelagens sísmicas, para cada seção de tiro comum presente no dado observado. São testadas quatro diferentes implementações: 1 GPU, 1 GPU + Multicore, 2 GPU e 2 GPU + Multicore, onde em cada rodada foram testados 9 (nove) diferentes *workloads* (aqui representado pelo número de tiros presente no dado observado).

Nossa implementação é testada no bem estabelecido modelo sintético de velocidades Marmousi, com 250 amostras verticais e 767 amostras horizontais. A geometria de aquisição consiste em 767 receptores, com tiros igualmente espaçados no eixo horizontal. São testadas 9 diferentes configurações de dado observado: 85, 170, 255, 340, 425, 510, 595, 680 e 765 tiros. Para cada uma dessas configurações, foram executadas 120 iterações da FWI. Os testes foram executados no sistema HETEROSOLAR, nó JUPITER, detalhado abaixo.

[Sistema 1 - JUPITER] Ambiente de execução com 2 GPUs. Possui 24 Intel Xeon com 2.00 GHz e 32 GB DDR3 em memória física, com dois *sockets* e 6 núcleos por *socket.* Possui duas GPUs NVIDIA Tesla C2075 com 14 *streaming multiprocessors* (SMs) e 32 *streaming processors* (SP) cada (448 cuda cores no total).

Cada uma das 9 configurações de tiros (e experimentos de FWI) consistem num *workload* total, que é adequadamente distribuído entre GPU (ou GPUs) e núcleos de CPU. Esta distribuição é baseada numa

adaptação de um esquema de *auto-tuning*[3]. A implementação é estática, e corresponde a uma única execução dos Passos 1 a 4, mostrados abaixo:

- **Passo 1:** alocação das variáveis necessárias, na GPU e CPU, com base na arquitetura do nó Jupiter.
- **Passo 2:** num esquema de implementação híbrido, o tempo do processador mais lento (no caso, os núcleos de CPU) para computar o máximo de *workload* que sua capacidade permite determinará quanto *workload* será dado ao(s) processador(es) mais rápido(s), de forma que ambos finalizem seus trabalhos simultaneamente. Para o nó Jupiter, esse tempo ***tcpu*** é dado pela computação de 23 tiros, um em cada núcleo de CPU. Daí, é medido o tempo de 1 tiro na GPU, ***t1gpu***. A razão de tempo ***tcpu/t1gpu*** é igual a 63, de forma que então os *workloads* 23 para *multicore* e 63 para a GPU garantem o melhor paralelismo. Esse processo é sumarizado pela primeira linha da Tabela 1, e define o *workload*-base de melhor paralelismo: 85 tiros (para a implementação 1GPU+Multicore).
- **Passo 3:** este passo avalia o *workload* total em comparação ao *workload*-base determinado pelo Passo 2; se o WL total for múltiplo do WL base, deve-se simplesmente executar o esquema o número necessário de vezes. Se não (sobra uma fração de tiros), esses tiros restantes são designados para a GPU.
- **Passo 4:** nessa adaptação, apenas uma iteração do esquema da Figura 1 é executada. O Passo 4 finaliza a etapa de configuração e distribuição, e marca o início da etapa de computação.

No caso da implementação 2GPU+Multicore, o processo é totalmente análogo, bastando tratar as duas GPU como uma única "super GPU" com o dobro da capacidade de processamento. Ao contrário dos recursos *multicore,* onde o tempo de processamento pode flutuar muito, no caso das GPUs o tempo de processamento de duas GPUs idênticas instaladas no mesmo nó é quase sempre o mesmo, com flutuações apenas na sexta casa decimal de segundos (para o nó Jupiter). A mesma análise é válida para três, quatro, etc, ou mais, GPUs. A Tabela 2 descreve o experimento com 2GPUs+Multicore, que é detalhado na próxima seção.

3. RESULTADOS E DISCUSSÃO

O primeiro grupo de testes foi executado para a implementação 1GPU+Multicore, como mostrado pela Tabela 1. Nesse grupo de testes, todos os *workloads* utilizados foram múltiplos exatos do WL base. A coluna 1 mostra o WL total; as colunas 2 e 3 mostram os WLs para GPU e núcleos da CPU, respectivamente. As colunas 4 e 5 mostram os tempos de processamento; para esta configuração, é esperado paralelismo máximo (ver **Passo 2** na seção anterior), o que não acontece exatamente, especialmente devido às flutuações nos tempos *multicore.* Mais iterações do esquema de *auto-tuning* mostrado na Figura 1 potencialmente reduziriam este problema, mas requer uma implementação mais sofisticada, que é objeto de trabalhos futuros. A última coluna na Tabela 1 mostra os *speedups* para cada WL total, calculados dividindo o tempo 1GPU+Multicore pelo tempo em 1 GPU. O *speedup* em média tem o valor de 1.37, ou 37% de ganho (Tabela 1).

O segundo grupo de testes foram feitos para a implementação 2GPU+Multicore, e é mostrado na Tabela 2. Neste grupo de testes os WL totais não são múltiplos exatos do WL base. O WL base nesta implementação corresponde a 150 tiros, sendo 122 feitos pelas 2 GPU e 22 para os núcleos de CPU. O fato do WL total não ser múltiplo do WL base prejudica a simultaneidade e paralelismo, pois a fração restante deve ser computada totalmente pelas GPU, gerando um tempo adicional com os recursos de CPU ociosos. Nesse esquema híbrido 2GPU+Multicore, quase todas as linhas (exceto a primeira) da Tabela 2, o esquema híbrido apresenta melhor performance que o esquema que utiliza apenas 2 GPUs. A tabela é análoga a Tabela 1, exceto pela coluna extra ao final, que mostra um segundo *speedup.* O primeiro *speedup* é calculado sobre o tempo da implementação com 1 GPU, enquanto que o segundo é feito sobre o tempo de 2 GPU (para o mesmo WL total). Analisando o *speedup* 1 é possível concluir que adicionar uma GPU ao sistema híbrido gera um aumento significativo de performance (cerca de 100%), e analisando o *speedup 2,* conclui-se que o sistema híbrido proporciona um ganho de cerca de 15% em relação ao sistema que utiliza apenas 2 GPUs.

Como observação final, o WL de 5 tiros para a CPU na linha 1 da Tabela 2 corresponde aa melhor tentativa de processar esses dados em *multicore* e 2 GPU. Foram testadas diversas configurações, por exemplo, 84 tiros para 2GPU, 1 tiro para a CPU (usando os 22 cores para acelerar esse único tiro), de forma que a melhor distribuição encontrada foi de 80 tiros para as 2 GPU e 5 tiros para os núcleos de CPU (4 *cores* por tiro). Mesmo assim, o tempo de computação utilizando apenas 2 GPU foi menor. Este resultado é

importante para mostrar que, abaixo do WL base (150, nesse caso), não vale a pena utilizar o esquema híbrido, pois os tempos de execução da CPU sempre degradarão o processo

Shots	GPU WL	CPU WL	t^{H}_{gpu} (10^3s)	t^{H}_{cpu} (10^3s)	t_{gpu} only (10^3s)	Speedup
85	62	23	35.26821	34.65084	48.99189	1.39
170	124	46	71.68966	68.32560	97.98379	1.37
255	186	69	107.52302	102.00036	146.97569	1.37
340	248	92	143.38225	134.69905	195.96758	1.37
425	310	115	178.58116	165.68958	244.95947	1.37
510	372	138	214.28372	199.36434	293.95138	1.37
595	434	161	250.00581	235.47930	342.94328	1.37
680	496	184	284.47366	279.64691	391.93516	1.38
765	558	207	334.56119	331.1007	440.92703	1.33

Shots	GPU WL	CPU WL	t^{H}_{gpu} (10^3s)	t^{H}_{cpu} (10^3s)	t_{gpu} only (10^3s)	t_{2gpu} only (10^3s)	Speedup 1	Speedup 2
85	80	5	26.49081	27.81828	48.99189	24.49595	1.85	0.92
170	148	22	42.62054	37.33506	97.98379	48.99189	2.30	1.15
255	233	22	67.37880	37.33506	146.97569	73.48784	2.18	1.09
340	296	44	85.23960	74.42610	195.96758	97.98379	2.30	1.15
425	381	44	109.97395	74.42610	244.96191	122.48095	2.23	1.11
510	444	66	127.85673	111.02910	293.95138	146.97569	2.30	1.15
595	529	66	152.61597	111.27312	342.94084	171.47042	2.25	1.12
680	592	88	170.46750	148.60819	391.93516	195.96758	2.30	1.15
765	655	110	188.75922	174.47430	440.92950	220.46475	2.34	1.17

Tabelas 1 e 2. Implementação 1GPU+Multicore (esquerda) e 2GPU+Multicore (direita).

4. CONSIDERAÇÕES FINAIS

Neste resumo expandido foram apresentadas algumas das implementações da FWI 2D mais utilizadas na literatura: multicore e multi-GPU. Foi apresentado um algoritmo híbrido, que faz uso de recursos multicore e multi-GPU simultaneamente, através de um esquema estático, onde a divisão de trabalho é feita adaptando-se um esquema de *auto-tuning*. É feita uma prova de conceito no bem estabelecido modelo de velocidades Marmousi. Resultados do algoritmo híbrido são comparados ao algoritmo multi-GPU, mostrando que houve *speedup* de 1.37 comparado ao algoritmo com 1 GPU, e de 1.15 comparado ao algoritmo com 2 GPUs. Os testes foram feitos num sistema com apenas 1 nó, ou seja, sem recursos multi-CPU. Os ganhos apresentados pelo algoritmo híbrido, apesar de modestos, indicam grande possibilidade de melhoria se forem utilizados recursos multi-CPU, o que pode ser explorado em trabalhos futuros. Sistemas heterogêneos mais complexos, como por exemplo multicore+multi-GPU+multi-MIC, com *clusters* e nós heterogêneos, que considerem realidade atual dos grandes centros de supercomputação, também devem ser testados.

Agradecimentos

Agradecemos ao Centro de Supercomputação SENAI-CIMATEC para Inovação Industrial, Empresa Brasileira de Inovação Industrial (EMBRAPII), Shell e ANP pelo financiamento desta pesquisa. Agradecemos também à Universidad de Murcia, por meio do *Parallel Computing Group of Universidad de Murcia*, pela permissão do uso do sistema HETEROSOLAR.

5. REFERÊNCIAS

[1] DOS SANTOS, Adriano. **Inversão de forma de onda aplicada a análise de velocidades sísmicas utilizando uma abordagem multiescala.** Dissertação de Mestrado, Universidade Federal da Bahia, Salvador, Brasil, 2013.

[2] YANG, P., GAO, J. e WANG, B. **A graphics processing unit implementation of time-domain full-waveform inversion.** Geophysics, 80(3), F31-F39, Tulsa, OK, Estados Unidos, 2015.

[3] BORATTO, M., ALONSO, P., GIMÉNEZ, D. e LASTOVETSKY, A. **Automatic tuning to performance modelling of matrix polynomials on multicore and multi-gpu systems.** The Journal of Supercomputing, 73(1), 227-239, Springer, Estados Unidos, 2017.

Declaração

Eu, **OSCAR FABIAN MOJICA LADINO**, orientador(a) de VICTOR KOEHNE RAMALHO, declaro ter realizado a análise e revisão do resumo expandido tendo como título: **“Modelo Computacional Heterogêneo para Inversão da Forma de Onda Completa 2D em Sistemas Multicore e MultiGPU”**.

Por ser verdade firmamos o presente.

Salvador, ___ de _____ de 2019.

_____

Oscar Fabian Mojica Ladino
Especialista III
SENAI CIMATEC

Declaração

Eu, **OSCAR FABIAN MOJICA LADINO**, orientador(a) de VICTOR KOEHNE RAMALHO, declaro ter realizado a análise e revisão do resumo expandido tendo como título: **"Modelo Computacional Heterogêneo para Inversão da Forma de Onda Completa 2D em Sistemas Multicore e MultiGPU"**.

Por ser verdade firmamos o presente.

Salvador, 10 de abril de 2019.

Oscar Mojica

Oscar Fabian Mojica Ladino
Especialista III
SENAI CIMATEC