Acelerando a Modelagem Sísmica Fletcher em GPUs por meio do uso de *Shared Memory*

Kenichi Brumati[1], Alexandre Sardinha[2],Alexandre da Silva Carissimi[1], Philippe O. A. Navaux[1], Arthur F. Lorenzon[1]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul – RS – Brasil

{kbrumati, asc, navaux, aflorenzon}@inf.ufrgs.br

[2]Petroleo Brasileiro S.A., Rio de Janeiro, Brazil

alexandre.sardinha@petrobras.com.br

Resumo. *A RTM (Reverse Time Migration) em meios TTI (Isotropia Transversal Inclinada) é fundamental para o cálculo sísmico de alta fidelidade na exploração de hidrocarbonetos. No entanto, a modelagem baseada em diferenças finitas apresenta alta complexidade computacional e é classicamente limitada pela movimentação de dados. Uma aplicação caracterizada por essa dificuldade é a aplicação Fletcher, uma simulação de propagação de ondas no tempo utilizando a discretização para cálculo de diferenças finitas, assim as modificações no código são focadas no controle explícito da hierarquia de memória. Este trabalho propõe utilizar técnicas de Shared Memory Tiling e reduzindo os acessos redundantes à memória global. Experimentos foram realizados nas GPUs NVIDIA RTX 4090 e L40S, a versão otimizada teve um ganho de desempenho em média de 17,2% em relação à versão base, sem o emprego de shared memory.*

1. Introdução

Na exploração de hidrocarbonetos, a RTM destaca-se como o padrão para a geração de imagens sísmicas de alta fidelidade. Sua precisão, no entanto, exige a modelagem da anisotropia TTI, frequentemente utilizando a formulação de *Fletcher* para diminuir instabilidades e artefatos de cisalhamento [Fletcher et al. 2009] . Porém, a resolução dessas equações gera *stencils* de alta ordem que tornam a aplicação classicamente limitada por memória, onde a movimentação de dados atua como o principal gargalo de desempenho.

Trabalhos recentes, como os de [Serpa et al. 2019], exploraram a eficiência deste método em arquiteturas *manycore* e aceleradores gráficos, estabelecendo implementações de referência focadas em paralelismo de dados e otimização de *cache*. Nesse mesmo contexto, tem-se investigado a portabilidade de desempenho da aplicação *Fletcher*, avaliando como manter a eficiência ao transitar por diferentes arquiteturas heterogêneas [Lorenzon et al. 2024]. Embora tais implementações apresentem ganhos expressivos de desempenho comparando diversas maneiras de paralelismo e abstrações de hardware, existe a possibilidade de explorar a hierarquia de memória para promover ainda mais otimizações. Uma delas é o uso de *shared memory* em GPUs baseada em técnicas de *tiling* [Micikevicius 2009].

Neste cenário, o presente trabalho tem como objetivo principal melhorar o desempenho da aplicação *Fletcher* através da redução da movimentação de dados entre os níveis de memória da GPU via o uso de *shared memory*. Diferentemente da versão base do

Fletcher, melhor detalhada na seção 2, que depende majoritariamente do gerenciamento implícito da hierarquia de *cache*, esta proposta visa explorar o controle programável da memória disponível em GPUs da NVIDIA. A motivação central encontra-se no carregamento cooperativo de blocos de dados para a memória compartilhada maximizando o reúso de dados no cálculo das diferenças finitas, atingindo assim um desempenho superior.

2. Modelagem *Fletcher*

A aplicação Modelagem Fletcher simula a propagação de ondas acústicas no domínio do tempo, reproduzindo a aquisição de dados em levantamentos sísmicos marinhos [Fletcher et al. 2009]. O modelo físico considera meios heterogêneos, onde a pressão $p(\mathbf{x}, t)$ é governada pela velocidade de propagação $V(\mathbf{x})$ e pela densidade $\rho(\mathbf{x})$.A solução numérica utiliza o método das diferenças finitas em grade regular. Nela, a discretização do estado futuro é calculado com base nos estados atuais.

A versão base, implementada por Jairo Panetta, utiliza uma abordagem direta em que as leituras dos campos de pressão P e Q são realizadas diretamente da memória global da GPU. Nesta abordagem, para cada ponto da grade (ix, iy) em uma iteração iz, o *stencil* requer o acesso aos vizinhos. Sem o uso explícito de cache gerenciado por software, isso resulta em acessos redundantes à memória global.

3. Otimizando o *Fletcher* com uso de *Shared Memory*

A versão modificada introduz otimizações focadas na reutilização de dados e no melhor uso de memória. As principais alterações algorítmicas implementadas nos *kernels* são:

Shared Memory Tiling: Foi utilizada a memória compartilhada para carregar planos 2D (xy) dos volumes de dados. Para cada passo em z (loop `iz`), um bloco de dados correspondente à dimensão do bloco de *threads*, acrescido da região de borda, é carregado cooperativamente para a memória compartilhada `s_pq`. Isso permite que as derivadas espaciais nas direções x e y (`Der1`, `Der2`) sejam calculadas acessando a memória compartilhada, que possui latência de ordens de magnitude inferiores à da memória global. **Carregamento em linha:** A transferência de dados da memória global para a compartilhada não depende estritamente da geometria dos *threads* de cálculo. Utilizou-se um loop linearizado com a variável iz para garantir que todos os *threads* do bloco participem carregando a *shared* e das bordas, maximizando a ocupação do barramento de memória [Liu et al. 2018]. **Reutilização de Registradores:** Após o carregamento, ponteiros para a memória compartilhada (`float *s_ptr`) são passados para as macros de derivadas, permitindo que o compilador otimize o acesso via registradores para os cálculos de estêncil locais.

4. Metodologia

Os experimentos foram conduzidos em duas máquinas do PCAD (Parque Computacional de Alto Desempenho), cujas configurações são as seguintes: Máquina Grace: 2 x Grace A02, 3.4 GHz, 144 *threads*, 144 cores, memória RAM de 480 GB LPDDR5, GPU NVIDIA L40S de 46 GB. Máquina Tupi: Intel(R) Xeon(R) E5-2620 v4, 2.1 GHz, 16 *threads*, 8 cores, memória RAM de 256 GB DDR4, GPU NVIDIA GeForce RTX 4090 24 GB. Ambas máquinas possuem sistema operacional DEBIAN 12. Os parâmetros de execução foram os seguintes: as dimensões do domínio computacional, configuração dos blocos de *threads*, compilador e número de execuções utilizadas nos testes foram respectivamente: *grid* (sx,sy,sz) computacional variada de 88 até 600, a cada execução o tamanho da *grid* é

incrementada em 32. Cada bloco de *threads* tem tamanho de (x,y) de (32,16). O compilador *NVCC* e *CUDA TOOLKIT* na versão 12.3. E por fim, cada experimento foi realizado 10 vezes.

5. Resultados Experimentais

A análise de desempenho adota como métrica principal a taxa de M*Samples*/s (Mega *Samples* por segundo). A progressão de desempenho para ambas as arquiteturas e abordagens de memória está ilustrada na Figura 1.

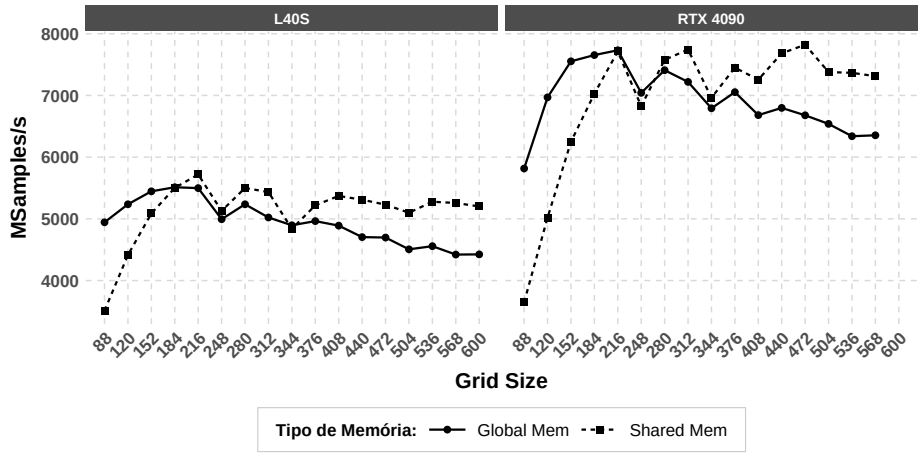
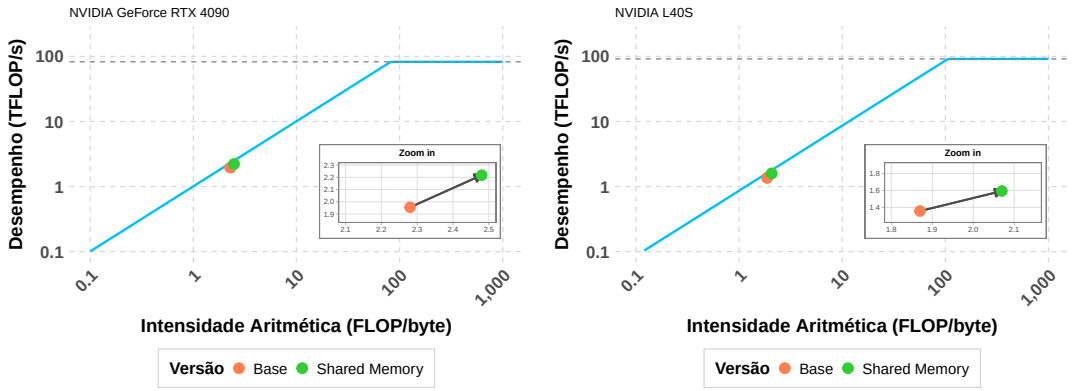


Figura 1. Resultados de comparação do uso de *Shared Memory*.

Com base nos dados apresentados nos gráficos, os experimentos demonstram que a versão otimizada obteve um ganho médio de 17,2% e desvio padrão de 1,08% em comparação com a versão base. Esse ganho se dá diante do uso da *shared memory* que tem latência inferior a memória global [Crovella 2022]. Outra parte importante é que existe um tamanho mínimo para que a melhoria seja perceptível, a partir desse ponto, quanto maior for a *grid*, maior é o pico de M*Samples*/s até atingir um determinado ponto de inflexão máximo, nos testes esse ponto se encontrou na *grid* de 472x472x472.

Para verificar a análise de desempenho no uso dos recursos de hardware e identificar possíveis fatores limitantes na execução, foi realizado uma avaliação *Roofline*. A Figura 2 apresenta essa análise focada no *kernel* de maior consumo computacional, *kernel* de propagação, mostrando a intensidade aritmética das operações (*TFlops/byte*) com o desempenho computacional alcançado (*TFlops/s*) para as GPUs RTX 4090 e L40S.

Diante da análise dos gráficos, mostra-se inicialmente que a versão base da simulação se posiciona em uma região limitada a memória, o que comprova a existência de um gargalo de acesso à memória global. O deslocamento positivo no gráfico em direção à direita demonstra que a versão otimizada com *Shared Memory* evidencia a redução desse gargalo, refletindo em uma melhora considerável no uso dos recursos de computação. Especificamente na GPU RTX 4090, a otimização fez com que a intensidade aritmética evoluísse de 2,28 para 2,48 *TFlops/byte*, acompanhada de um aumento no desempenho computacional de 1,9549 para 2,2171 *TFlops/s* (Figura 2-a). Um comportamento similar foi registrado na GPU L40S, que apresentou um salto na intensidade aritmética de 1,87 para 2,07 *TFlops/byte* e um crescimento na taxa de processamento de 1,3573 para 1,5935 *TFlops/s* (Figura 2-b).



(a) *Roofline* 4090 (b) *Roofline* L40S

Figura 2. Roofline do kernel de maior consumo computacional

6. Conclusão

Este trabalho apresentou uma otimização eficiente para a simulação de ondas em meios TTI, mitigando a limitação clássica de acesso à memória. Utilizando Shared Memory Tiling e reúso de registradores, substituindo os acessos redundantes à memória global por carregamentos cooperativos de blocos de dados. Experimentos em GPUs NVIDIA RTX 4090 e L40S demonstraram a eficácia da proposta, com um ganho médio de 17,2% no desempenho.

Agradecimentos

Os autores gostariam de agradecer à Petrobras pelo apoio financeiro. O presente trabalho contou com o apoio da infraestrutura do PCAD para a realização dos testes experimentais.

Referências

Crovella, R. (2022). Re: Why reg to shared to global is faster than reg to global? NVIDIA Developer Forums. Acessado em: 24 de fevereiro de 2026.

Fletcher, R. P., Du, X., and Fowler, P. J. (2009). Reverse time migration in tilted transversely isotropic (TTI) media. *Geophysics*, 74(6).

Liu, C., Wang, Q., Chu, X., and Leung, Y.-W. (2018). *IEEE Transactions on Parallel and Distributed Systems*, PP:1–1.

Lorenzon, A. F., Navaux, P. O. A., Sardinha, A., and Messer, B. (2024). Towards performance portability of an oil and gas application on heterogeneous architectures. In *2024 IEEE 36th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 127–137.

Micikevicius, P. (2009). In *3D finite difference computation on GPUs using CUDA. Proceedings of the 2nd Workshop on General Purpose Processing on Graphics Processing Units (GPGPU-2)*, Washington, D.C., USA. ACM.

Serpa, M., Pavan, P., Panetta, J., Azambuja, A., Carissimi, A., and Navaux, P. (2019). In *Portabilidade e Eficiência do Método Fletcher de Aplicações Sísmicas em Arquiteturas Multicore e GPU. Anais do XX Simpósio em Sistemas Computacionais de Alto Desempenho*, Porto Alegre, RS, Brasil. SBC.