Otimização de Configurações de Threads por Bloco para Kernels GPU na Exploração Geofísica *

Brenda S. Schussler[1], **Pedro H. C. Rigon**[1], **Cristiano A. Künas,** [1]
Arthur F. Lorenzon[1], **Alexandre Carissimi**[1], **Philippe O. A. Navaux**[1]

[1]Instituto de Informática – UFRGS - Porto Alegre – RS – Brazil

{bsschussler, phcrigon, cakunas, aflorenzon, asc, navaux}@inf.ufrgs.br

Resumo. *A busca por desempenho e eficiência energética em aplicações de exploração geofísica impulsiona a otimização dos recursos de hardware, especialmente em GPUs. Nessas arquiteturas, a configuração de execução de cada kernel é crucial, influenciando diretamente no desempenho e consumo de energia. Nesta pesquisa, propomos uma estratégia exaustiva de otimização de threads por bloco em tempo de execução com salvamento em disco, obtendo ganhos médios superiores a 20% no EDP (produto energia-tempo) em relação às execuções padrão do Fletcher.*

1. Introdução

Os métodos de exploração geofísica tornaram-se essenciais na atualidade, pois auxiliam na descoberta de recursos fundamentais como o petróleo e o gás. Em contrapartida, a preservação do meio ambiente torna-se uma questão essencial, e portanto, uma limitação à expansão dos esforços de exploração petrolífera. Neste contexto, aplicações de simulação sísmica tem sido utilizadas para reduzir os impactos ambientais e melhorar a precisão de perfuração, como por exemplo, a modelagem *Fletcher*, base deste trabalho. Uma vez que estes modelos são altamente paralelos, são ideais para serem computados em unidades de processamento gráfico (GPUs).

No entanto, as GPUs demandam uma potência significativa para o seu funcionamento. Assim, para obter o melhor desempenho e mitigar o consumo energético é essencial a otimização dos recursos de hardware disponíveis nas GPUs [Navaux et al. 2023]. Nesse sentido, uma das principais maneiras de otimizar o uso de recursos das GPUs é através do uso da configuração de execução mais adequada do *kernel* a ser processado. No entanto, encontrar as configurações ideais que proporcionem o melhor desempenho e minimizem o consumo de energia é uma tarefa complexa dada a quantidade de variáveis envolvidas (e.g., número de kernels, conjunto de entrada e microarquitetura) [Schussler et al. 2023].

Neste cenário, este trabalho propõe um algoritmo que explora o espaço de possibilidades de configuração do kernel em tempo de execução. O algoritmo não necessita nenhuma informação específica da aplicação para convergir à configuração ideal de número de blocos e *threads* por bloco que apresenta o melhor EDP (produto energia-tempo) para a modelagem *Fletcher*. Esta métrica é utilizada com o intuito de encontrar o melhor custo-benefício considerando o desempenho e o consumo energético. Com a aplicação do nosso

*Este estudo foi parcialmente apoiado pela Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001, pela Petrobras sob número 2020/00182-5 e pelo edital CNPq/MCTI/FNDCT - Universal 18/2021 sob número 406182/2021-3. Alguns experimentos deste trabalho utilizaram os recursos da infraestrutura PCAD, http://gppd-hpc.inf.ufrgs.br, no INF/UFRGS.

algoritmo, obtivemos ganhos de EDP superiores a 60% em alguns cenários específicos, como por exemplo para o tamanho de grade 184. Além disso, analisando o cenário geral da aplicação, obtivemos um ganho médio de 22% para a comparação com o *baseline* 16×16 e de 11% para o *baseline* 32×16.

2. Modelagem Fletcher

O método de *Fletcher* é uma técnica utilizada no campo da geofísica para simular a coleta de dados em levantamentos sísmicos para obter informações detalhadas sobre diferentes estruturas geológicas e potenciais reservatórios de petróleo [Fletcher et al. 2009].

A implementação CUDA do método *Fletcher* utiliza uma grade tridimensional para representar a propagação da onda no ambiente. As dimensões dessa estrutura são definidas pelas entradas da aplicação que se referem ao tamanho dos eixos *x*, *y* e *z* da grade. Para que esta grade seja computada por CUDA *threads* em paralelo durante a execução da aplicação, é utilizada uma abordagem de decomposição 2D do domínio. Isto é, obtém-se um plano bidimensional através de um corte no volume da grade (por exemplo, nas dimensões *x* e *y*). Então, o algoritmo pode iterar ao longo da terceira direção, neste caso, representada pela dimensão *z*. Isso permite que este plano bidimensional (*x,y*) seja dividido em blocos de CUDA *threads*, os quais possuem um determinado número de threads nas dimensões x e y. No algoritmo de *Fletcher*, o programador deve definir o número de *threads* por bloco nas dimensões *x* e *y* ao lançar um *kernel* para execução na GPU.

3. Aplicação da Busca Exaustiva com Salvamento em Disco

A nossa implementação da busca exaustiva com salvamento em disco inicialmente verifica a existência de um arquivo com a configuração ideal salvo em disco, levando em consideração a aplicação em execução e o tamanho da grade de entrada. Isso torna-se particularmente relevante, uma vez que para cada versão da aplicação, arquitetura utilizada e dimensão de grade de entrada diferentes, uma configuração distinta pode ser a ideal. Caso o arquivo contenha a configuração ideal para o conjunto de entrada em questão, a aplicação é executada com as configurações ideais já definidas.

Algorithm 1 Algoritmo de Exploração do Espaço de Configurações do *kernel*

Entrada: Tamanho da grade em que a onda será propagada
Saída: Configuração ideal do Kernel CUDA para esta entrada
Verificar existência de arquivo de configuração salvo em disco
if Arquivo de configuração encontrado **then**
    Carregar configuração do disco
else
    Iniciar busca exaustiva
    for cada configuração possível do Kernel CUDA **do**
        Coletar métricas de desempenho e energia
    end for
    Identificar melhor configuração
    Continuar a execução da aplicação utilizando a melhor configuração
    Salvar configuração otimizada em disco
end if
Utilizar configuração otimizada nas execuções subsequentes

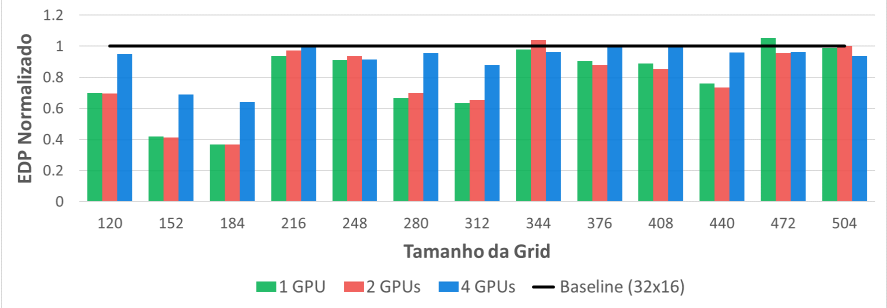
Por outro lado, não havendo um arquivo de configuração previamente salvo, o algoritmo inicia uma busca exaustiva, de modo a garantir que todo o espaço de combinações seja devidamente explorado, iterando sobre o *kernel* CUDA testando diferentes configurações. O tempo de execução e a potência consumida pelas GPUs na

execução destes *kernels* são coletados e avaliados, repetindo-se este processo até que todo o conjunto de configurações seja explorado. Definida a configuração que resulta no melhor EDP (produto energia x tempo de execução) para aquele contexto, a aplicação segue a execução utilizando esta configuração até concluir e então faz o salvamento desta configuração em disco. Deste modo, nas execuções subsequentes nesta arquitetura, o algoritmo utilizará diretamente essa configuração otimizada, economizando recursos computacionais ao evitar uma nova busca exaustiva. Essa implementação é exemplificada no Algoritmo 1.

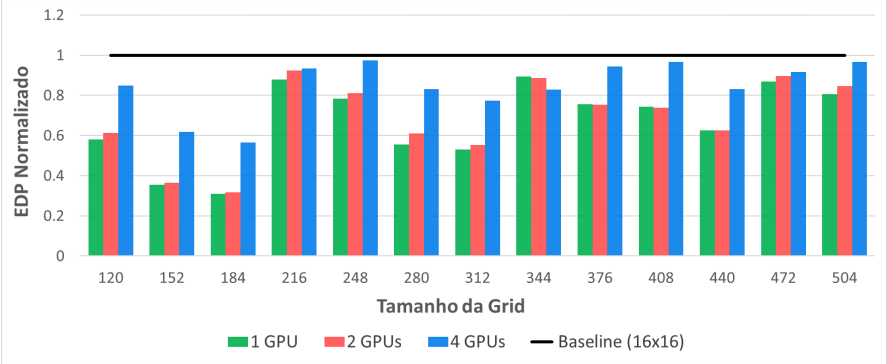
4. Discussão dos Resultados

Os experimentos foram realizados utilizando as versões CUDA do método *Fletcher* para 1 GPU, 2 GPUs e 4 GPUs [Rigon et al. 2023]. Cada versão foi executada 10 vezes em 15 grades de tamanho crescente. As execuções foram realizadas no Parque Computacional de Alto Desempenho da UFRGS (PCAD) utilizando uma arquitetura Pascal com 4 GPUs NVIDIA Tesla P100-SXM2-16GB, com 3584 núcleos CUDA e dois processadores Intel Xeon.

O termo EDP diz respeito ao produto entre energia e tempo de execução, portanto, é ideal quando busca-se um equilíbrio entre desempenho e consumo energético. A Figura 1 exibe um comparativo dos resultados obtidos aplicando-se o método de exploração exaustiva com dois casos de *baseline* diferentes: *Plataforma Padrão*, onde os kernels são executados com o número de *threads* por bloco sugerido pelo fabricante da GPU, por exemplo, a NVIDIA sugere a configuração 16×16 [NVIDIA 2023]; e *Padrão Fletcher*,



(a) Configuração *Baseline Fletcher* 32x16



(b) Configuração *Baseline* CUDA 16x16

Figura 1. EDP do nosso algoritmo nas versões *Single* GPU, 2 GPUs e 4 GPUs em comparação com as configuração *baseline*, calculado para 13 tamanhos de grade [120-504].

utilizando a configuração padrão do método *Fletcher*, definida pelos desenvolvedores da aplicação - 32×16 *threads* por bloco. Os resultados estão normalizados pelos respectivos *baselines* considerados - representados por uma linha preta. Portanto, valores abaixo da linha (y=1.0) indicam que a busca exaustiva produziu resultados melhores do que a estratégia comparada.

Analisando os gráficos percebe-se que a busca exaustiva obtém ganhos consideráveis de EDP em relação à ambos os *baselines*. Em comparação à configuração *baseline* CUDA 16x16 (Figura 1(b)), foi obtido um ganho de EDP médio de 22%, enquanto para a configuração *baseline Fletcher* 32x16 (Figura 1(a)), o ganho médio foi de $\approx 10\%$.

Em geral as buscas exaustivas são demoradas, pois demandam um considerável tempo para explorar as possibilidades existentes. No entanto, como nossa abordagem é implementada em tempo de execução, as diferentes configurações dos *kernels* são testadas ao longo da execução do programa enquanto o espaço de possibilidades ainda não foi integralmente explorado. Considerando o fato de que simulações sísmicas requerem um número elevado de iterações, o tempo da busca não torna-se impactante para a aplicação e ao fim compensa, pois finalizada a busca, a aplicação segue executando com a configuração ideal até o fim de suas iterações.

5. Conclusão

O método de exploração exaustiva resultou em ganhos significativos de EDP em comparação com os padrões de execução *baseline*. Além disso, nossa implementação em tempo de execução permite que o *overhead* da busca exaustiva seja dissolvido ao longo das iterações da aplicação. Nesse contexto de simulações geofísicas que requerem um número de iterações bastante alto, o tempo da busca torna-se irrelevante ao final da execução, conseguindo obter ganhos de desempenho da aplicação mesmo com sua implementação.

Finalmente, é possível concluir que o algoritmo proposto mostrou-se altamente eficiente, mesmo sendo uma busca exaustiva. Ademais, em situações em que as configurações ideais já estão salvas em disco, o ganho de EDP da aplicação será ainda maior, pois não será necessário executar iterações com configurações não ótimas, que geram uma pequena degradação no EDP, apesar de ainda assim o resultado final ser vantajoso, como mostrado nesta pesquisa.

Referências

Fletcher, R. P., Du, X., and Fowler, P. J. (2009). Reverse time migration in tilted transversely isotropic (TTI) media. *Geophysics*, 74(6):WCA179–WCA187.

Navaux, P. O. A., Lorenzon, A. F., and da Silva Serpa, M. (2023). Challenges in high-performance computing. *Journal of the Brazilian Computer Society*, 29(1):51–62.

NVIDIA, G. D. (2023). CUDA C++ Best Practices Guide.

Rigon, P. H., Schussler, B. S., Padoin, E. L., Lorenzon, A. F., Carissimi, A., and Navaux, P. O. (2023). Towards a Multi-GPU Implementation of a Seismic Application. In *Latin American High Performance Computing Conference*, pages 146–159. Springer.

Schussler, B. S., Rigon, P. H., Lorenzon, A. F., Carissimi, A., and Navaux, P. O. (2023). The Impact of CUDA Execution Configuration Parameters on the Performance and Energy of a Seismic Application. In *Latin American High Performance Computing Conference*, pages 170–183. Springer.