COMPARAÇÃO DE BENCHMARKS ENTRE LINGUAGENS DE DOMÍNIO ESPECÍFICO PARA GPUS DE PROPÓSITO GERAL

GABRIEL MOURA DE SOUSA GUILHERME[1]; ANDRE RAUBER DU BOIS[2]

[1]*Universidade Federal de Pelotas – gmsguilherme@inf.ufpel.edu.br*
[2]*Universidade Federal de Pelotas – dubois@inf.ufpel.edu.br*

1. INTRODUÇÃO

As Unidades de Processamento Gráficos (GPUs) são cada vez mais usadas para tarefas de propósito geral, com destaque para a *NVIDIA's Compute Unified Device Architecture* (CUDA), um dos paradigmas para computação em GPU com forte presença em diversos setores da indústria nas últimas décadas (SANDERS; KANDROT, 2011).

Contudo, o desenvolvimento de software para GPUs de propósito geral ainda é majoritariamente feito em linguagens de programação de baixo nível, como a própria linguagem CUDA. O desenvolvimento de software em linguagens de baixo nível pode ser um desafio em diversas situações, o que estimulou o desenvolvimento de novas abordagens para agilizar o processo. Muitas dessas técnicas se baseiam no uso de funções de alta ordem, um conceito trazido do paradigma funcional de programação, como esqueletos algorítmicos que envolvem padrões usados na programação paralela como um todo (COLE, 1989).

Visando abordar estas questões, foi desenvolvida a PolyHok, uma Linguagem de Domínio Específico ou *Domain-Specific Language* (DSL) que implementa kernels polimórficos de ordem superior para a GPU. PolyHok desenvolvida como uma extensão da linguagem de programação Elixir, uma linguagem de programação funcional de alto nível conhecida por sua capacidade de performance em computação paralela e distribuída (THOMAS, 2018). Este artigo foca nas comparações de performance entre a execução de programas desenvolvidos em PolyHok e seus equivalentes tanto em outras DSLs quanto em CUDA puro.

2. METODOLOGIA

Para o desenvolvimento deste trabalho, a implementação oficial da PolyHok foi obtida de seu repositório oficial, em conjunto com a implementação da GPotion, uma DSL desenvolvida anteriormente com os mesmos propósitos da PolyHok. Para que as comparações pudessem ser feitas, algumas modificações foram feitas no código-fonte original. Novos benchmarks foram desenvolvidos em PolyHok com base nas suas implementações equivalentes para a GPotion.

Todas as duas versões dos benchmarks apresentam, ao final da execução dos algoritmos, o tempo de execução em milissegundos (ms). Os algoritmos utilizados para comparação foram: Reduce, uma operação que busca reduzir os elementos de uma sequência em um único valor, e Matrix Multiplication (MM), que multiplica matrizes de elementos float. Os experimentos foram realizados em uma máquina com as seguintes especificações: Processador do modelo AMD Ryzen 7 9700X com 8 cores de processamento, 128GB de memória RAM e duas GPUs NVIDIA GeForce RTX 4070. Nas duas DSLs, os benchmarks foram executados

com o comando: mix run benchmarks/*nome_do_benchmark.ex* 1024, sendo "benchmarks o repositório onde se encontram os arquivos de teste.

3. RESULTADOS E DISCUSSÃO

Todos os testes foram executados com as mesmas entradas. A PolyHok apresentou um tempo de resposta maior que a GPotion, ainda que com valores semelhantes. Para o algoritmo Reduce, PolyHok apresentou uma resposta de 227ms, enquanto que o tempo de resposta da GPotion foi de 172ms. Já para o algoritmo MM, a PolyHok demorou um total de 219ms para execução, enquanto que a GPotion demorou um total de 182ms. A comparação dos resultados pode ser vista no gráfico a seguir.

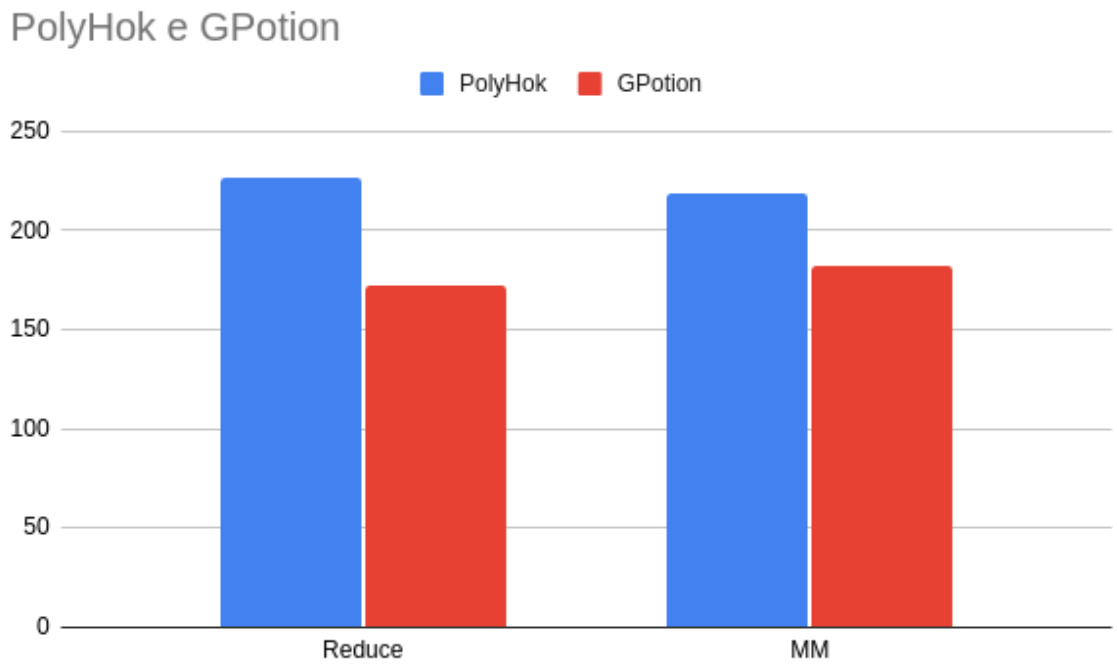

Figura 1: Gráfico de Comparação Entre PolyHok e GPotion.

4. CONCLUSÕES

Como foi observado, apesar da PolyHok ser uma DSL mais recente e de possuir mais recursos que sua antecessora, a GPotion ainda apresenta resultados melhores nos benchmarks. Como trabalhos futuros, há espaço para testes com benchmarks diferentes e modificações no código-fonte da PolyHok que visem uma melhora de desempenho.

5. REFERÊNCIAS BIBLIOGRÁFICAS

Livro
SOBRENOME, Letras Iniciais dos Nomes. **Título do Livro**. Local de Edição: Editora, ano da publicação.
Ex.: JENNINGS, P.B. **The practice of large animal surgery**. Philadelphia: Saunders, 1985. 2v.

SANDERS, J; KANDROT, E. **CUDA By Example: An Introduction to General-Purpose GPU Programming**. Addison-Wesley, 2011. Cap 1, p.8-11

THOMAS, D. **Programming Elixir >= 1.6: Functional |> Concurrent |> Pragmatic |> Fun**. The Pragmatic Programmers, 2018.

COLE, Murray I. **Algorithmic skeletons: structured management of parallel computation**. London: Pitman, 1989.