Explorando a DSL PolyHok no desenvolvimento de programas para GPU

RENZO ROSSATO BATTISTI[1]; ANDRE RAUBER DU BOIS[2]

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

1. INTRODUÇÃO

As placas de vídeo (ou GPUs – graphics processing unit) surgiram nos anos 70-80, inicialmente com o intuito de uso exclusivo a processamento gráfico. Hoje em dia, Nvidia, a maior empresa fabricante de GPUs, é a empresa mais valiosa do mundo, demonstrando a importância delas no mundo atual.

No entanto, elas sempre foram notoriamente difíceis de programar para computação de propósito geral (isto é, computação não necessariamente de processamento gráfico). Isso dificultou o uso dela para áreas de pesquisa, com pesquisadores tendo que utilizar APIs de processamento gráfico para resolver problemas gerais, enganando a GPU ao fingir que seus problemas eram de renderização (SANDERS; KANDROT, 2010). Com a criação da API de baixo nível CUDA, as placas de vídeo se tornaram um pouco mais acessíveis, e seu uso em áreas de pesquisa cresceu.

Em contraste, processadores tradicionais (*CPUs*) vêm, há décadas, sendo beneficiados por linguagens de programação e bibliotecas que abstraem grande parte da complexidade de seu funcionamento interno. A falta de um grau similar de abstração na programação de GPUs motiva a criação de ferramentas que reduzam a barreira de entrada para desenvolvedores.

A DSL para elixir, PolyHok (DU BOIS; CAVALHEIRO, 2025), pretende amenizar este problema. Ela permite a criação de kernels GPU polimórficos de alta ordem, isto é, kernels de GPU que podem aceitar funções de dispositivo, incluindo funções anônimas, como argumentos que são tipados dinâmicamente em runtime baseado nos argumentos que eles recebem.

No entanto, essas facilidades podem vir com overhead. O objetivo deste trabalho é, portanto, traduzir programas usados em benchmarks CUDA para programas equivalentes em PolyHok, à propósito de comparar os custos computacionais. Isso permitirá um entendimento essencial dos custos envolvidos em usar mais abstrações na programação para placas de vídeo, permitindo aos desenvolvedores fazerem uma análise informada dos custos/benefícios, auxiliando-os a escolherem a melhor ferramenta para cada caso.

2. METODOLOGIA

Como metodologia, foram escolhidos alguns programas implementados em CUDA do benchmark PolyBench/GPU 1.0 (Scott Grauer-Gray et al., 2012). Estes então foram traduzidos manualmente para PolyHok. Para verificar que o funcionamento dos programas era idêntico, os resultados obtidos pelos programas, tanto em CUDA quanto em PolyHok, são armazenados em arquivos, e depois comparados com um script em C. A tradução é considerada correta então quando todos os números de saída são iguais em várias configurações iniciais simultaneamente.

Os programas podem ser facilmente adaptados para aplicar seus respectivos algoritmos em uma grande ou pequena quantidade de dados. Foram priorizados números mais altos, pois o caso para se usar PolyHok é mais forte quando o overhead é distribuído entre um tempo maior de computação.

Cada configuração de tamanho foi executada diversas vezes para reduzir o impacto de variações ocasionais no tempo de execução. Especificamente, para cada programa, foram feitas entre 30 e 40 execuções de cada versão (CUDA e PolyHok), calculando-se a média dos tempos obtidos. Esse processo foi repetido para três diferentes tamanhos de entrada por programa, resultando em série de dados que podem ser usados para comparar o desempenho entre as versões.

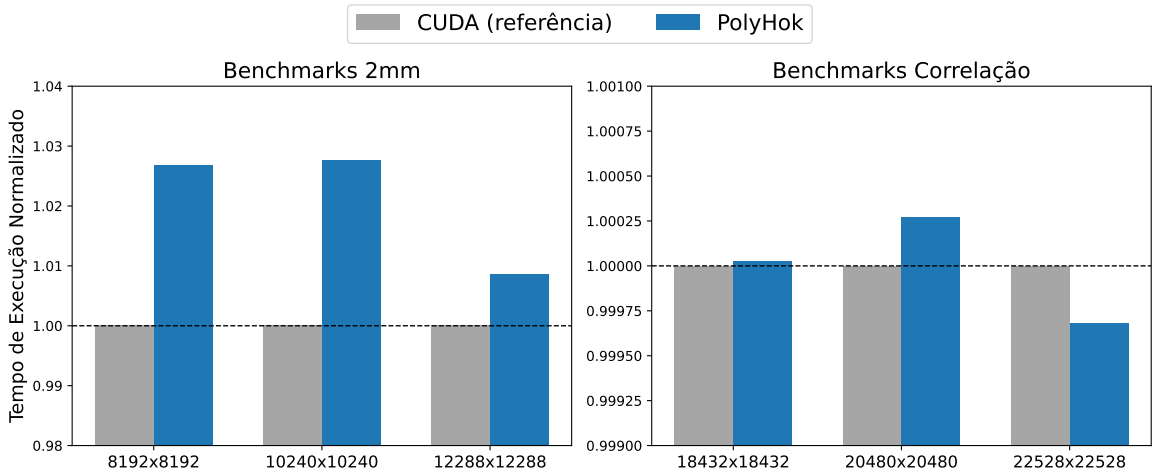
3. RESULTADOS E DISCUSSÃO

Até o momento, foram traduzidos manualmente dois programas com o objetivo de benchmarking: o 2mm, que executa duas multiplicações de matrizes em sequência, e o Correlação, que calcula a correlação entre m variáveis com n amostras. Os gráficos abaixo demonstram a difereça entre os tempos de execução obtidos, que foram normalizados usando o tempo da versão CUDA como referência. Note que o eixo Y é diferente entre os gráficos.

Nos testes com o 2mm, o tempo de execução do PolyHok foi 2,68% maior que o do CUDA para matrizes de ordem 8192, 2,76% maior para matrizes de ordem 10240 e 0,85% maior para matrizes de ordem 12280.

No programa de correlação, o tempo de execução do PolyHok foi 0,0028% maior que o do CUDA para matrizes de ordem 18432, 0,027% maior para matrizes de ordem 20480, enquanto para matrizes de ordem 22528 foi 0,031% menor. Este último resultado demonstra que o overhead que PolyHok possui em relação a CUDA é pequeno o suficiente para em alguns casos não ser detectável.

De maneira geral, os resultados demonstram que o *overhead* de PolyHok em relação ao CUDA é consistentemente baixo, situando-se em menos de 3% na maioria dos casos. Isso sugere que a adoção de uma linguagem mais expressiva e com maior abstração, como a PolyHok, pode ser viável mesmo em cenários que precisam de desempenho, em especial nos casos que o custo computacional seja elevado o suficiente para tornar o overhead comparativamente insignficante.



4. CONCLUSÕES

Este trabalho apresentou uma análise comparativa do desempenho entre implementações em CUDA e implementações equivalentes em PolyHok, visando compreender o impacto de se utilizar uma ferramenta de programação para GPU com maior nível de abstração.

Os resultados obtidos indicam que, embora a PolyHok introduza um pequeno overhead em relação ao CUDA, esse custo é, na maioria dos casos, inferior a 3% e pode ser desprezível para entradas muito grandes. Esse achado é relevante para desenvolvedores e pesquisadores, pois demonstra que é possível obter ganhos significativos em produtividade e legibilidade do código sem perdas substanciais de desempenho.

Com isso, a PolyHok se apresenta como uma alternativa promissora para a programação de GPUs, especialmente em contextos onde a rapidez no desenvolvimento, a manutenibilidade e a clareza do código sejam prioridades.

Esperamos que este trabalho seja um passo para tornar a programação de GPUs uma tecnologia mais simples e produtiva, demonstrando que níveis mais altos de abstração podem coexistir com eficiência computacional.

5. REFERÊNCIAS BIBLIOGRÁFICAS

DU BOIS, A.; CAVALHEIRO, G. Polymorphic higher-order gpu kernels. In: 31ST INTER-NATIONAL SYMPOSIUM ON PARALLEL AND DISTRIBUTED COMPUTING (EUROPAR 2025), 2025.

SANDERS, J.; KANDROT, E. CUDA by example: an introduction to general-purpose GPU programming. Boston: Addison-Wesley, 2010. 1 v.

GRAUER-GRAY, S.; XU, L.; SEARLES, R.; AYALASOMAYAJULA, S.; CAVAZOS, J. Auto-tuning a high-level language targeted to GPU codes. In: INNOVATIVE PARALLEL COMPUTING (INPAR), 1., San Jose, 2012. Anais... San Jose: Institute of Electrical and Electronics Engineers, 2012. p. 1-10.