Proposta de *Pipelines* Lineares de Alto Nível em Rust Utilizando GPU

Leonardo G. Faé[1], **Dalvan Griebler**[1]

[1] Escola Politécnica, Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Porto Alegre – RS – Brasil

leonardo.fae@edu.pucrs.br,dalvan.griebler@pucrs.br

Resumo. *Unidades de Processamento Gráfico (GPUs) são unidades de hardware projetadas para processar quantidades massivas de dados em paralelo. Rust é uma nova linguagem de programação de baixo nível com foco em desempenho e segurança. Até o momento, há poucos trabalhos acadêmicos sobre abstrações de alto nível para GPUs em Rust. Propomos uma possível abstração, baseada no padrão de pipeline e implementada utilizando macros procedurais.*

1. Introdução

Unidades de Processamento Gráfico (GPUs) possuem uma arquitetura de *hardware* especializada para computações massivamente paralelas [NVIDIA 2023]. Elas são comumente utilizadas para acelerar aplicações científicas [Kondratyuk et al. 2021] e modelos atuais de treinamento e inferência em inteligência artificial [Cabodi et al. 2019]. GPUs são geralmente programadas em CUDA [NVIDIA 2023] ou OpenCL [Khronos 2023], embora haja esforços para permitir abstrações de programação de alto nível em muitas linguagens de programação (por exemplo, [Du Bois and Cavalheiro 2023]).

Rust [The Rust Project 2023] é uma linguagem de programação recente com destaque nos últimos anos, sendo um marco de seu sucesso sua inclusão no Kernel do Linux[1]. Rust tem um foco explícito em desempenho e segurança, sendo uma linguagem de baixo nível com análise estática robusta que garante que todo programa Rust, ao compilar, não contém comportamento indefinido (*Undefined Behavior*).

Como Rust é concebida como uma linguagem de programação de baixo nível, sua comunidade tem se dedicado para transcrever abstrações e bibliotecas criadas para C/C++ para Rust. Isso também é válido para programação em GPU, onde projetos como Rust-GPU[2] tentam compilar Rust diretamente para PTX, o alvo de compilação de CUDA. Infelizmente, isso significa que houve pouco trabalho dedicado a oferecer abstrações de alto nível para programação em GPU em Rust. Gostaríamos de propor uma abstração, que fará uso de macros em Rust para implementar uma interface fácil de usar, permitindo que o programador invoque a GPU em um *pipeline* de processamento.

2. Proposta de Pesquisa

Nossa proposta de abstração é baseada em macros procedurais Rust, a estratégia utilizada também em um trabalho anterior com foco em arquiteturas *multi-core* [Faé et al. 2023]. Neste trabalho, adotamos um abordagem diferente, como pode ser visto no Código 1. A

[1] *Git Pull*: `https://lore.kernel.org/lkml/202210010816.1317F2C@keescook/`
[2] Disponível em: `https://github.com/Rust-GPU/Rust-CUDA`

notação `#[GPU]` acima da função é o que Rust chama de macro procedural de atributo. Macros em Rust são diferentes de linguagens como C/C++, onde são meras substituições de texto. Em Rust, quando o compilador encontra uma macro, ele chama um programa separado, pequeno, definido pelo programador, e alimenta-o com o código ao qual a macro se refere como entrada. Neste caso, a entrada será a declaração e definição da função `gpu_computation`. Nesse pequeno programa, devemos implementar toda a lógica de transformação de código para converter o código Rust em código executável em GPU. Nossos alvos atuais para geração de código são CUDA e OpenCL.

```
#[GPU]
fn gpu_computation(input: SomeStruct) -> OutputStruct {
    // codigo que consome SomeStruct e gera OutputStruct usando a GPU
}
```

Código 1. Exemplo do objetivo da proposta

A abstração proposta não poderia ser implementada para todo código Rust válido. Por exemplo, é impossível traduzir o *hash map* da biblioteca padrão de Rust para CUDA ou OpenCL. Atualmente, esperamos ser capazes de transformar código que opera exclusivamente em tipos primitivos, ou em *arrays* e *slices* de tipos primitivos. É possível que também possamos trabalhar com *structs* simples, desde que estejam anotadas com `#[repr(C)]`; em outras palavras, desde que tenham uma representação em memória equivalente à de C. Além disso, estamos planejando que o código da Listagem 1 seja facilmente chamado de dentro de um *pipeline* de processamento, explorando um padrão paralelo simples e de fácil compreensão [Mattson et al. 2004].

Esta será a primeira abstração de alto nível para programação em GPU em Rust na literatura acadêmica. Trabalhos futuros incluiriam investigar se poderíamos fazê-lo funcionar com construções de programação Rust mais complexas, e talvez com padrões paralelos diferentes além de *pipelines*.

Referências

Cabodi, G., Camurati, P., Garbo, A., Giorelli, M., Quer, S., and Savarese, F. (2019). A smart many-core implementation of a motion planning framework along a reference path for autonomous cars. *Electronics*, 8(2).

Du Bois, A. R. and Cavalheiro, G. (2023). Gpotion: An embedded dsl for gpu programming in elixir. In *Proceedings of the XXVII Brazilian Symposium on Programming Languages*, SBLP '23, New York, NY, USA. Association for Computing Machinery.

Faé, L. G., Hoffman, R. B., and Griebler, D. (2023). Source-to-source code transformation on rust for high-level stream parallelism. In *Proceedings of the XXVII Brazilian Symposium on Programming Languages*, SBLP '23, page 41–49, New York, NY, USA. Association for Computing Machinery.

Khronos (2023). The OpenCL Specification.

Kondratyuk, N., Nikolskiy, V., Pavlov, D., and Stegailov, V. (2021). Gpu-accelerated molecular dynamics: State-of-art software performance and porting from nvidia cuda to amd hip. *The International Journal of High Performance Computing Applications*, 35(4):312–324.

Mattson, T., Sanders, B., and Massingill, B. (2004). *Patterns for Parallel Programming*. Addison-Wesley Professional, first edition.

NVIDIA (2023). *CUDA C++ Programming Guide*. NVIDIA.

The Rust Project (2023). The Rust Reference.