Estendendo o RustStreamBench com Renoir e novos *benchmarks*

Lucas S. Bianchessi[1], **Leonardo G. Faé**[1], **Dalvan Griebler**[1]

[1]Politécnica – Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brasil

l.bianchessi@edu.pucrs.br, leonardo.fae@edu.pucrs.br

Resumo. *Neste trabalho, foram adicionados ao RustStreamBench dois novos benchmarks e uma biblioteca de paralelismo nova, o Renoir. Renoir apesar de atingir um desempenho semelhante com bibliotecas de paralelismo mais consolidadas, a biblioteca não é capaz de paralelizar o benchmark image-processing, por demandar memória demais, um erro semelhante à versão distribuída da biblioteca que estourava a pilha independente do benchmark.*

1. Introdução

Rust é uma linguagem de programação oficialmente publicada em 2015 pela *Mozilla Research* [Matsakis and Klock 2014], tendo como promessa a programação de aplicações seguras e rápidas a partir de seus conceitos de posse (tradução livre de *ownership*) e tempo de vida (*lifetime*), que são regras de segurança exigidas pelo compilador. A posse é a garantia que existirá apenas uma variável que é dona do dado e quando sair do escopo o dado será limpo, enquanto o tempo de vida de uma variável assegura que uma variável não existirá para além do escopo em que é utilizada. Essas regras, além de outras escolhas específicas de implementação, fazem Rust não ter comportamento indefinido.

Com o número de dados que se coleta crescendo desamparadamente, [Statista 2020]. Aplicações de *stream* processam continuamente os dados na medida em que são produzidos. Essas aplicações muitas vezes devem fazer uso de processamento paralelo para cumprir com o volume de dados que devem ser processados.

1.1. Paralelismo em Rust

Atualmente, o ecossistema de paralelismo em Rust é dominado por duas bibliotecas amplamente adotadas pela comunidade: Rayon, especializada em paralelismo de dados, e Tokio, voltada para execução assíncrona. Ambas se destacam por oferecer alto desempenho e excelente programabilidade, fruto de anos de otimização e contribuições de inúmeros desenvolvedores. Esse amadurecimento contínuo permitiu que Rayon e Tokio superassem outras bibliotecas em termos de eficiência e facilidade de uso, consolidando-se como as principais ferramentas para paralelismo em Rust.

Além dessas bibliotecas consolidadas, este estudo também inclui a Spar-Rust([Faé et al. 2023]) uma biblioteca focada em paralelismo de *stream*. Originalmente desenvolvida em C++, a Spar foi traduzida para Rust, mantendo sua proposta de oferecer suporte a processamento paralelo de fluxos de dados. A escolha da Spar-Rust para este trabalho se deve justamente ao seu foco específico em paralelismo de *stream*, uma área que complementa as capacidades oferecidas pelo Rayon e pelo Tokio, permitindo uma análise mais abrangente das soluções de paralelismo disponíveis em Rust.

Renoir é uma biblioteca de paralelismo em Rust, publicada em [De Martini et al. 2024], projetada especificamente para paralelismo de *stream*. Suas funções simulam o comportamento dos iteradores do Rust, inspiradas no paradigma de Rayon, fundamental para a programação nesta linguagem. Uma das principais promessas de Renoir, além do desempenho, é sua flexibilidade, oferecendo suporte tanto para paralelismo de memória compartilhada quanto para sistemas distribuídos, tornando-a uma ferramenta versátil para diferentes cenários de computação paralela.

1.2. RustStreamBench

Originalmente publicada em [Pieper et al. 2021] "RustStreamBench" é um conjunto de *benchmarks* voltados para paralelismo de *stream*, continha em seu lançamento quatro *benchmarks* distintos, o Bzip2, compressão e descompressão de arquivos, image-processing, aplicar diversos filtros em sequência em determinadas imagens, eye-detector, detecta faces e os olhos dentro dessas faces no vídeo providenciado, e o micro-bench uma pseudo-aplicação foi retirada do presente artigo por ser considerada uma aplicação artificial demais.

2. Trabalhos Relacionados

Rust, por ser uma linguagem de programação relativamente recente, continua em processo de consolidação na comunidade científica. No entanto, estudos como [Costanzo et al. 2021] demonstram que Rust possui potencial para se tornar uma linguagem de alto desempenho, o artigo compara tanto a programabilidade quanto a performance das linguagens, e em relação ao desempenho, C performa melhor, enquanto Rust foi considerado com maior programabilidade e segurança.

Diversos *benchmarks* já foram realizados para avaliar a eficácia de Rust em cenários de alto desempenho. Por exemplo, [Ivanov 2022] compara algoritmos de ordenação em várias linguagens, destacando a competitividade da eficiência de Rust. Esses estudos são fundamentais para expandir o conhecimento sobre as capacidades de Rust em aplicações de alto desempenho.

Este artigo se diferencia dos trabalhos mencionados por focar especificamente em *benchmarks* de paralelismo de *stream*, uma área ainda pouco explorada no ecossistema Rust. Ao analisar o desempenho de Rust nesse contexto, buscamos contribuir para o entendimento de sua viabilidade em cenários que exigem processamento paralelo e distribuído de fluxos de dados. A maior contribuição seria a adição de dois *benchmarks*, o kmeans e o word-count, ao *RustStreamBench* e também a uma nova biblioteca de paralelismo, o Renoir.

3. Experimentos

Os experimentos foram realizados em uma máquina com duas CPUs Intel(R) Xeon(R) Gold 5118 @ 2.30GHz, 190GB de RAM, com Ubuntu 20.04.6. Os gráficos da Figura 1 demonstram o desempenho em segundos em relação ao número de *threads*. As bibliotecas de paralelismo utilizadas foram o Renoir, Rayon, Spar-rust, Tokio e Standard-threads (paralelismo nativo de Rust). Renoir foi removido do *benchmark* "image-processing" devido ao erro de falta de memória (Out Of Memory).

Verificamos que o Renoir perde performance nas aplicações Bzip2 e face detector na medida em que aumentamos o número de *threads*. Além disso, não escala bem na aplicação de word-count. O Rayon tem boa escalabilidade em todas as aplicações, com o seu *speedup* diminuindo quando chegamos no *hyperthreading* com mais de 20 *threads* de execução. Por outro lado, Tokio apresenta dificuldade no word-count, pois não é uma biblioteca apropriada para o paralelismo de dados demandado pela aplicação. Já o Spar-Rust apresenta dificuldade em executar tanto o word-count quanto o kmeans. Finalmente, o alto desvio padrão de Std-Threads no word-count se explica pela aplicação se beneficiar bastante de técnicas de balanceamento de carga, o que Std-Threads não faz.

4. Conclusão

Contando que o *benchmark* image-processing não pôde ser executado devido a erros de "out of memory" e que toda a versão do Renoir distribuído não funcionou devido a *stack overflow*, fica claro a ineficiência de uso de memória pela biblioteca. Apesar disto, Renoir atingiu, nos *benchmarks* um nível de desempenho comparável ao de bibliotecas já estabelecidas no ecossistema Rust, como Rayon e Tokio, o que, por si só, já representa um mérito significativo. Renoir foi adicionado parcialmente com sucesso no *RustStramBench* devido à falta de capacidade da própria biblioteca, para trabalhos futuros deverá ser acompanhado o desenvolvimento de novas bibliotecas de paralelismo, principalmente as de *stream* para serem adicionadas no *RustStreamBench* e estudadas mais a fundo[1].

References

Costanzo, M., Rucci, E., Naiouf, M., and Giusti, A. D. (2021). Performance vs programming effort between rust and c on multicore architectures: Case study in n-body. In *2021 XLVII Latin American Computing Conference (CLEI)*, pages 1–10.

De Martini, L., Margara, A., Cugola, G., Donadoni, M., and Morassutto, E. (2024). The renoir dataflow platform: Efficient data processing without complexity. *Future Generation Computer Systems*, 160:472–488.

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.

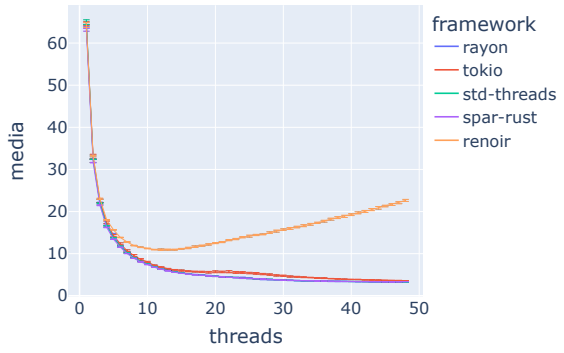
Ivanov, N. (2022). Is rust c++-fast? benchmarking system languages on everyday routines.

Matsakis, N. D. and Klock, F. S. (2014). The rust language. In *Proceedings of the 2014 ACM SIGAda Annual Conference on High Integrity Language Technology*, HILT '14, page 103–104, New York, NY, USA. Association for Computing Machinery.

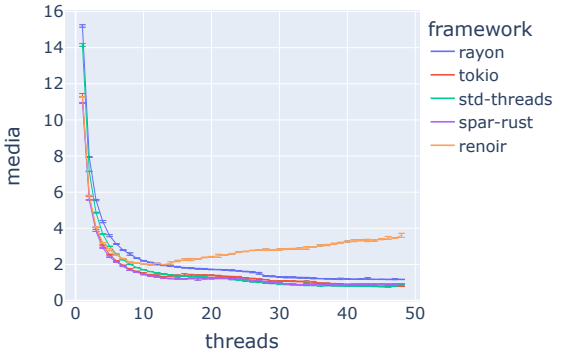
Pieper, R., Löff, J., Hoffmann, R. B., Griebler, D., and Fernandes, L. G. (2021). High-level and efficient structured stream parallelism for rust on multi-cores. *Journal of Computer Languages*, 65:101054.

Statista (2020). Statista. [Online]. Available: `https://www.statista.com/statistics/871513/worldwide-data-created/`.

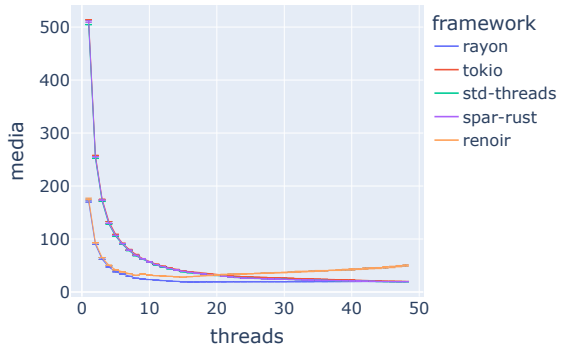
[1]Esta pesquisa foi financiada com o apoio de FAPERGS 09/2023 PqG (Nº 24/2551-0001400-4) e CNPq Research Program (Nº306511/2021-5)



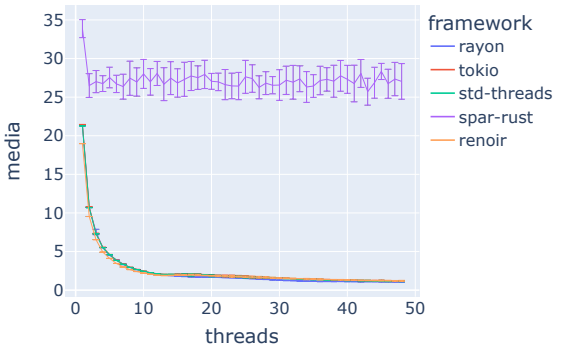
(a) `bzip2` compressão



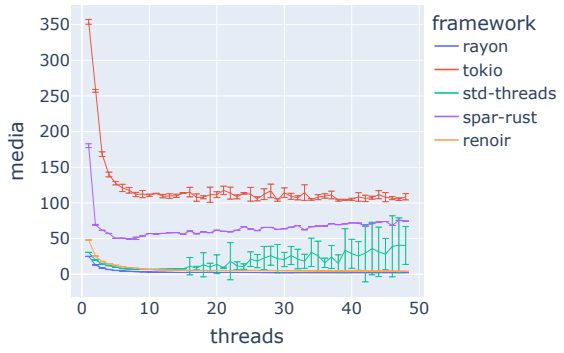
(b) `bzip2` descompressão



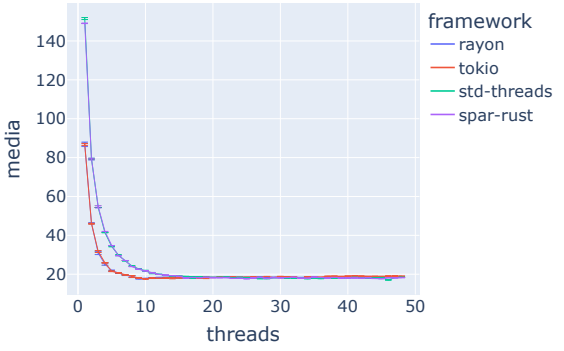
(c) `Face-detector`



(d) `kmeans`



(e) `word-count`



(f) `image-processing`

Figure 1. Resultados