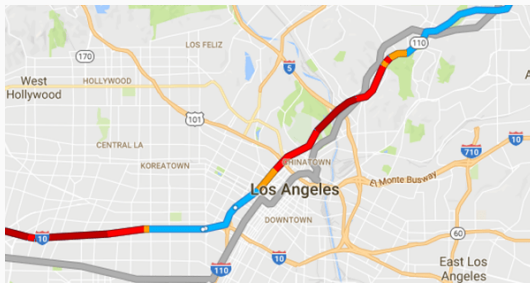


Eliminando Gargalos de Processamento Utilizando Rust

Johnathan Fercher

1. Introdução
2. Quem usa em produção?
3. Programando em Rust
4. Resolução de um problema

Introdução



- Um gargalo é a parte menos eficiente de um sistema:
 - 90% de um trajeto é feito a 110 km/h e 10% é feito a 20 km/h;
 - Um caixa 24 horas dentro de uma loja que fecha;

- IO-Bound;
- CPU-Bound;

- CPU-Bound:
 - Utiliza todo o processamento, porém, ainda demora;

- CPU-Bound:
 - → Algorithms, Parallel Programming, Programming Languages;

- CPU-Bound:
 - → Algorithms, Parallel Programming, Programming Languages;



- Programming Language;
- Memory and thread-safe;
- Rust \rightarrow LLVM \rightarrow EXE;
- C-Bindings;
- Object-Oriented and Functional;
- Unit-tests and Package Manager;
- Interfaces and Generics;
- Without Garbage Collector;

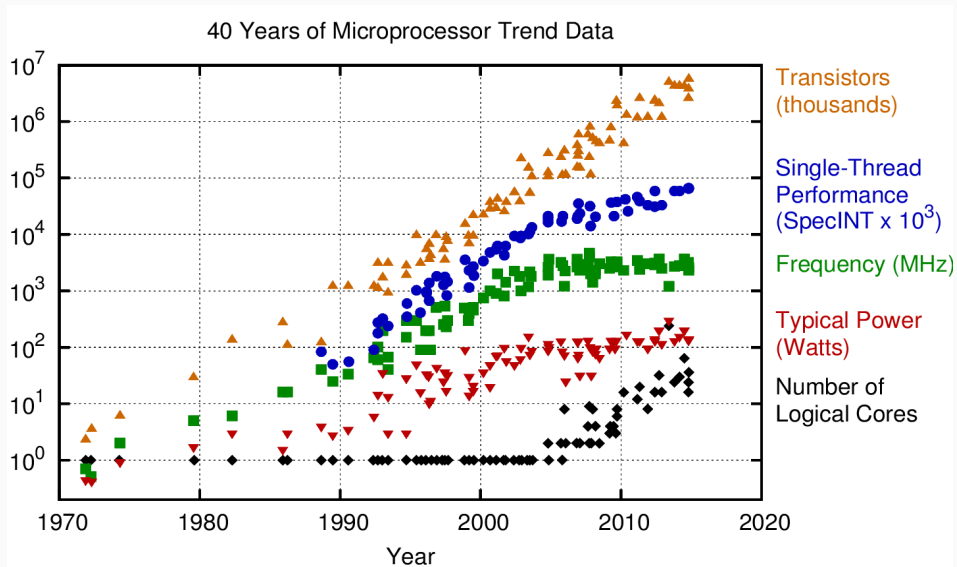
"O clock dos processadores dobra a cada 18 meses."

Lei de Moore, 1965.

Motivação (Acha que isso ainda funciona?)



Motivação



"The way the processor industry is going, is to add more and more cores, but nobody knows how to program those things. I mean, two, yeah; four, not really; eight, forget it."

Steve Jobs, Apple.

Bug 650064

Running Aurora and Firefox in parallel

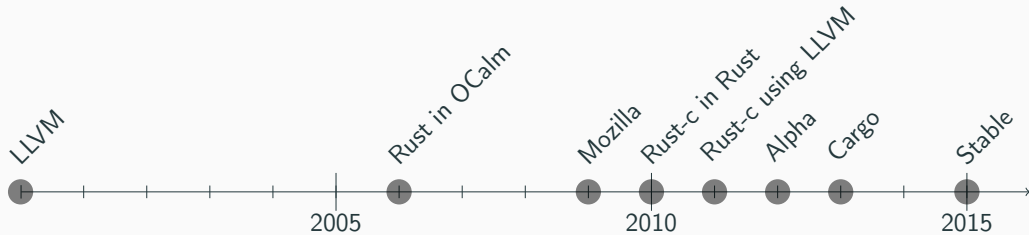
UNCONFIRMED Unassigned

▼ Status

Product: Firefox ▼
Component: General ▼
Status: UNCONFIRMED

Reported: 8 years ago
Modified: 6 years ago

História



- Licença MIT no Github;
- Duas versões: Stable e Nightly;
- Atualizações a cada 6 semanas;
- Processo de RFC;
- Quando uma RFC é aprovada ela é adicionada na versão Nightly;
- Após algum tempo em Nightly, ela pode ser adicionada na versão Stable, deixada de lado ou alterada;

Provide a split method which doesn't consume the element used to split #53890

 Open johnfercher opened this issue on Sep 1 · 0 comments



johnfercher commented on Sep 1 · edited ▼



The `split` method consumes the element used to split, how you can see in the code below. But in some cases would be nice to move the element to the "first or second slice" like the `split_at` method does, but to `n` slices.

The numpy library has a `method` like that.

<https://play.rust-lang.org/?gist=94dd413054604e4efffa96a5e8ed72b5&version=stable&mode=debug&edition=2015>

Code

```
#[derive(Clone, Debug)]
struct Delta {
    pub val: i32
}
```

Quem usa em produção?

- Friends of Rust · The Rust Programming Language:
 - <https://www.rust-lang.org/pt-BR/friends.html>

Programando em Rust

Hello World

- cargo new nome_do_projeto --bin
- cargo run
- cargo test
- cargo run --release

```
fn main() {  
    println!("Hello World");  
}
```

Mutabilidade x Imutabilidade

```
let foo = vec![1, 2, 3, 4];  
foo.push(5); // Não compila
```

```
let mut bar = vec![1, 2, 3, 4];  
bar.push(5); // Compila
```

Thread-safe

```
let numbers = vec![1, 2, 3, 4];  
  
for i in 0..10 {  
    thread.spawn(|| {  
        // Não compila  
        let mut t_numbers = &numbers;  
  
        for number in t_numbers {  
            // Do something  
        }  
  
        t_numbers.clean();  
    });  
}
```

3 Formas de Filtrar uma Lista

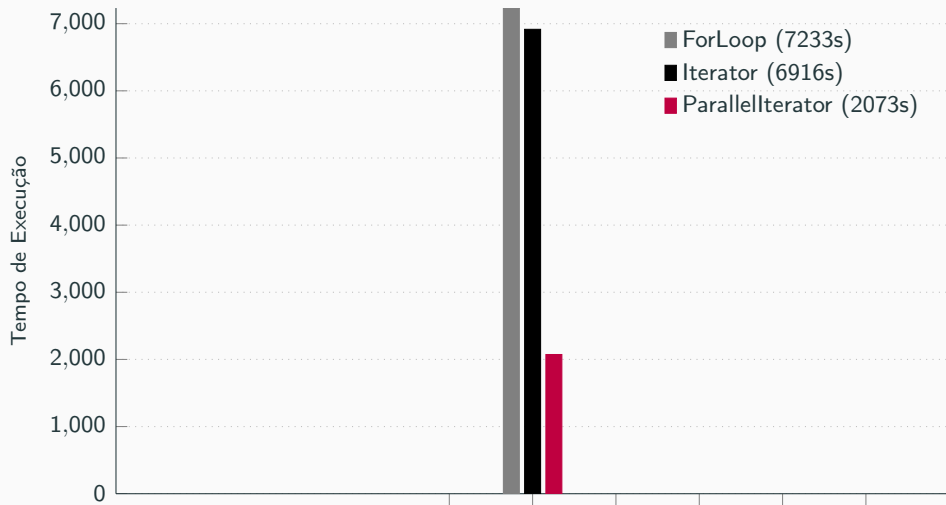
```
let mut primes = Vec::new();

for number in numbers {
    if is_prime(number) {
        primes.push(number);
    }
}
```

```
let primes = numbers.iter()
    .filter(|x| is_prime(x))
    .collect::<Vec<u64>>();
```

```
let primes = numbers.par_iter()
    .filter(|x| is_prime(x))
    .collect::<Vec<u64>>();
```


Benchmark Para 150.000 Números



Limitando o Paralelismo

```
let pool = ThreadPoolBuilder::new()
    .num_threads(6)
    .stack_size(4_000_000)
    .build()?;

pool.install(|| {
    let primes = numbers.par_iter()
        .filter(|x| is_prime(x))
        .collect::<Vec<u64>>();
});
```



Cargo.TOML

```
[package]
name = "velocity_simulator"
version = "0.1.0"
authors = ["John Fercher <johnathanfercher22@gmail.com>"]

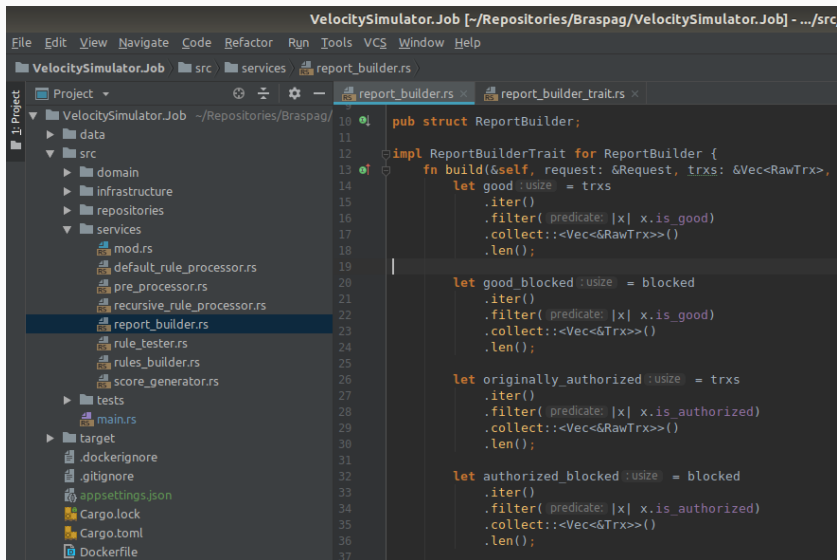
[dependencies]
rayon = "1.0.2"
```

Testes unitários

```
#[test]
fn when_add_should_add_correctly() {
    let calculator = Calculator::new();
    let added = calculator.add(10, 10)

    assert_eq!(added, 20);
}
```

IDEs: JetBrains, Visual Studio Code, ...



Dockerfile

```
# Define imagem de compilação
FROM yasuyuky/rust-ssl-static as build

COPY ./ ./

# Atualiza versão do Rust e adiciona target musl
RUN rustup install stable
RUN rustup default stable
RUN rustup target add x86_64-unknown-linux-musl
RUN rustup update

# Instala dependências
ENV DEBIAN_FRONTEND=noninteractive
RUN apt-get update && apt-get -y install ca-certificates libssl-dev && rm -rf /var/lib/apt/lists/*

# Habilita cross-compile
ENV PKG_CONFIG_ALLOW_CROSS=1

# Compila projeto otimizado para a arquitetura musl
RUN cargo build --target x86_64-unknown-linux-musl --release

# Cria pasta de execução
RUN mkdir -p /build-out

# Copia eo executável
RUN cp target/x86_64-unknown-linux-musl/release/velocity_simulator /build-out/
```


```
# Define imagem de execução
FROM scratch

# Copia certificados
COPY --from=build /etc/ssl/certs/ca-certificates.crt /etc/ssl/certs/ca-certificates.crt

# Copia executável
COPY --from=build /build-out/velocity_simulator /


# Define variáveis de ambiente
ENV SSL_CERT_FILE=/etc/ssl/certs/ca-certificates.crt
ENV SSL_CERT_DIR=/etc/ssl/certs

# Comando de execução
CMD ["/velocity_simulator"]
```

Dashboard

PRIVATE REPOSITORY

johnfercher/velocity-simulator 

Last pushed: a month ago

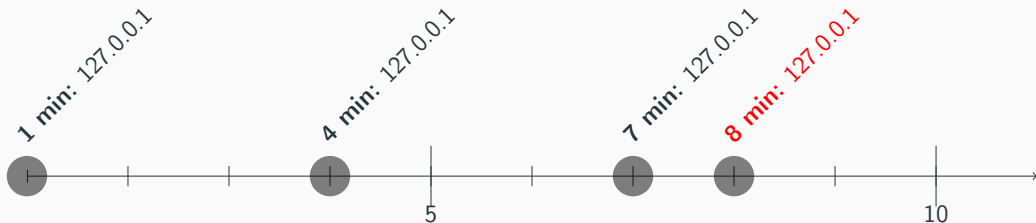
[Repo Info](#) [Tags](#) [Collaborators](#) [Webhooks](#) [Settings](#)

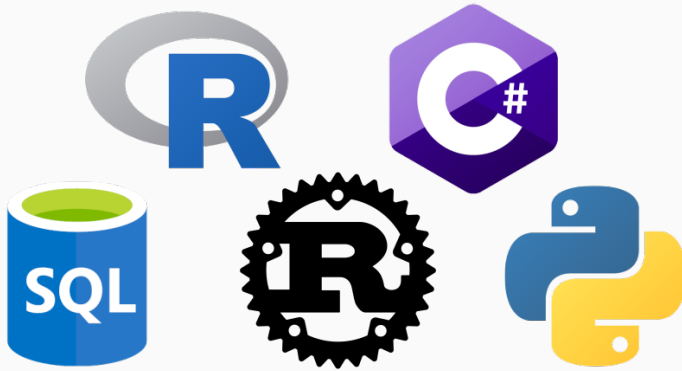
Tag Name	Compressed Size	Last Updated
latest	4 MB	a month ago

Resolução de um problema

Regras de repetição:

- 5 repetições de um **Cpf** em 10 minutos;
- 10 repetições de um **Cartão** em 2 dias;
- 2 repetições de um **Ip** em 5 minutos;

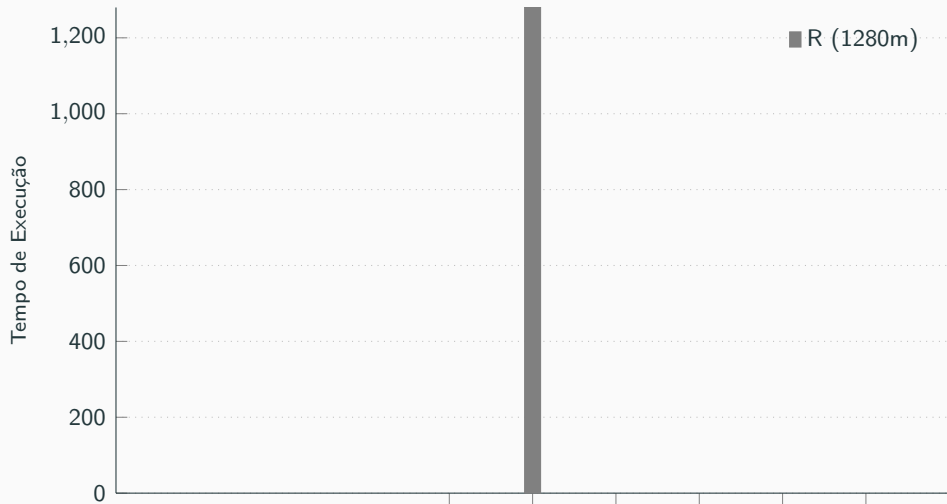




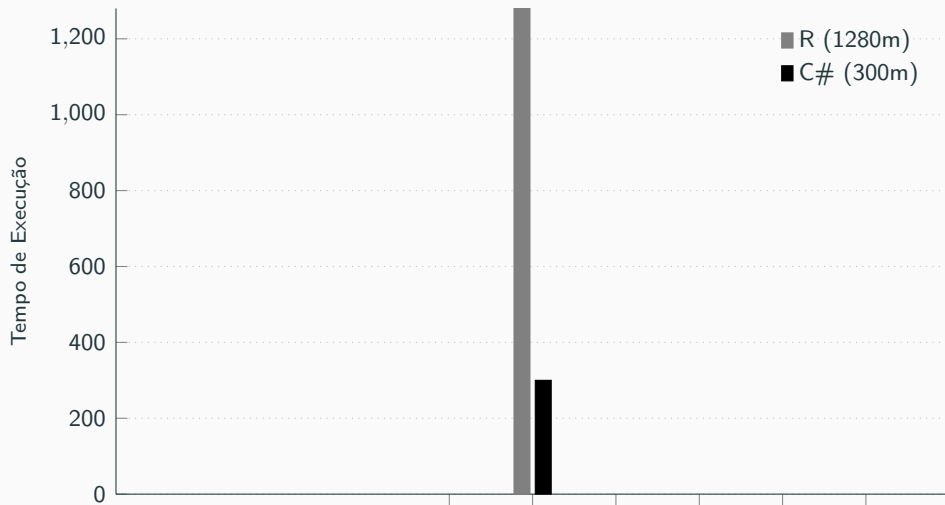
Hardware: I7 8770 (6 núcleos + 6 threads), 8GB RAM DDR4, SSD;

- Não é utilizado nenhuma técnica de agrupamento;
- Não é utilizado nenhum artifício de programação funcional;

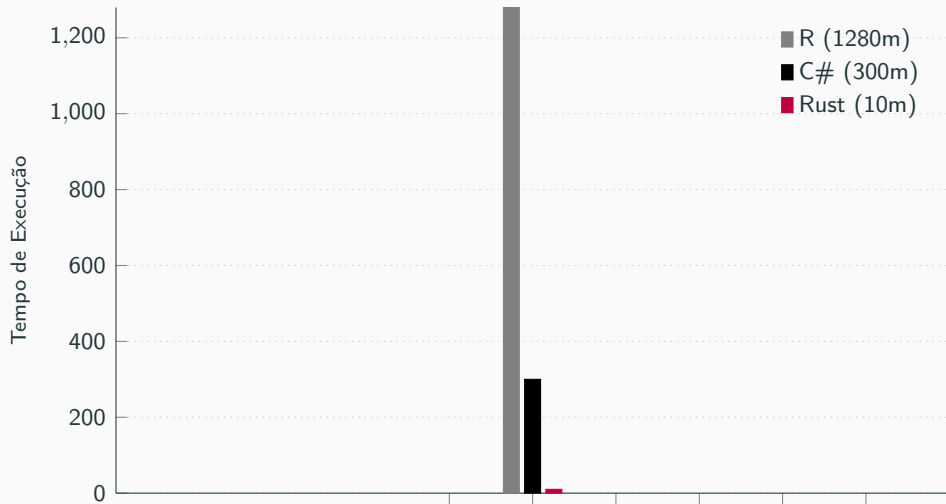
Benchmark (v1.0)



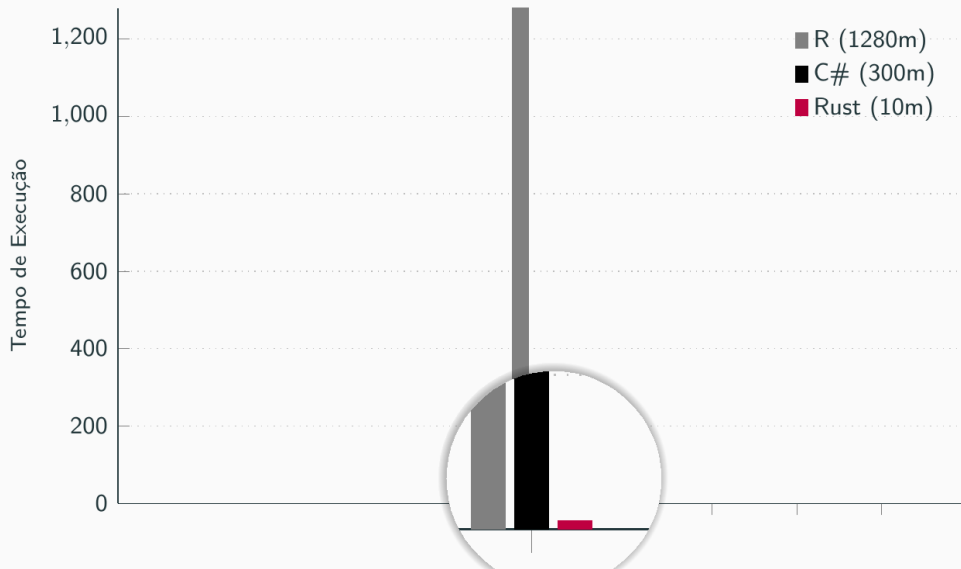
Benchmark (v1.0)



Benchmark (v1.0)



Benchmark (v1.0)

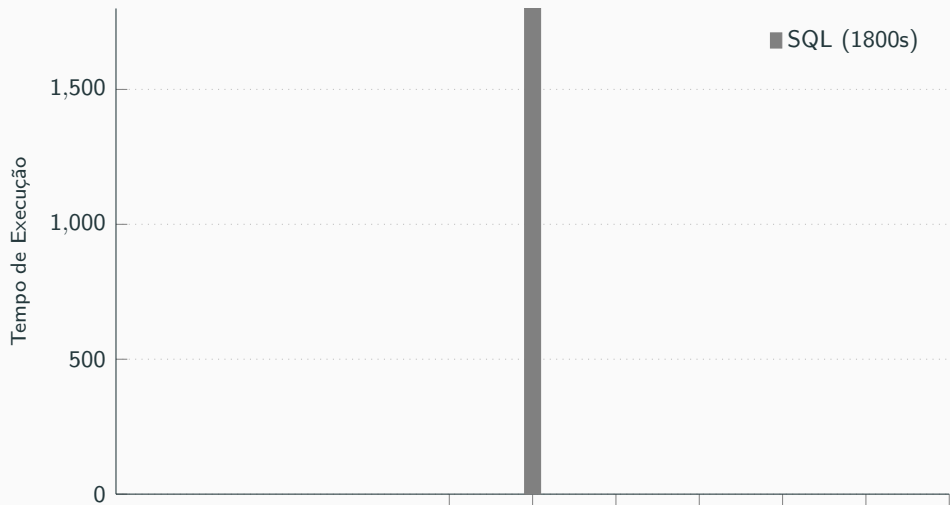




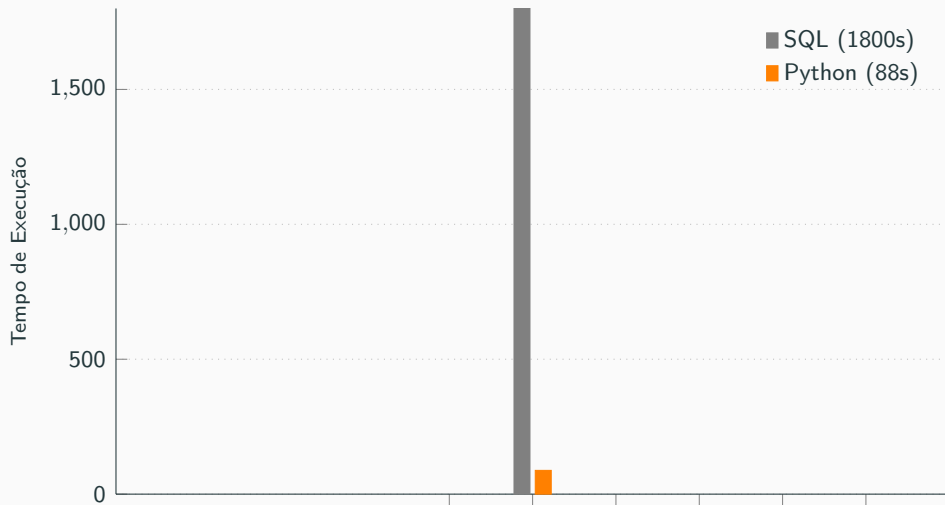
Hardware: I7 8770 (6 núcleos + 6 threads), 8GB RAM DDR4, SSD;

- Todos os algoritmos utilizam agrupamento;
- Rust e Python utilizam programação funcional;
- Python utiliza a lib Numpy, que é feita em C, C++ e Fortran;

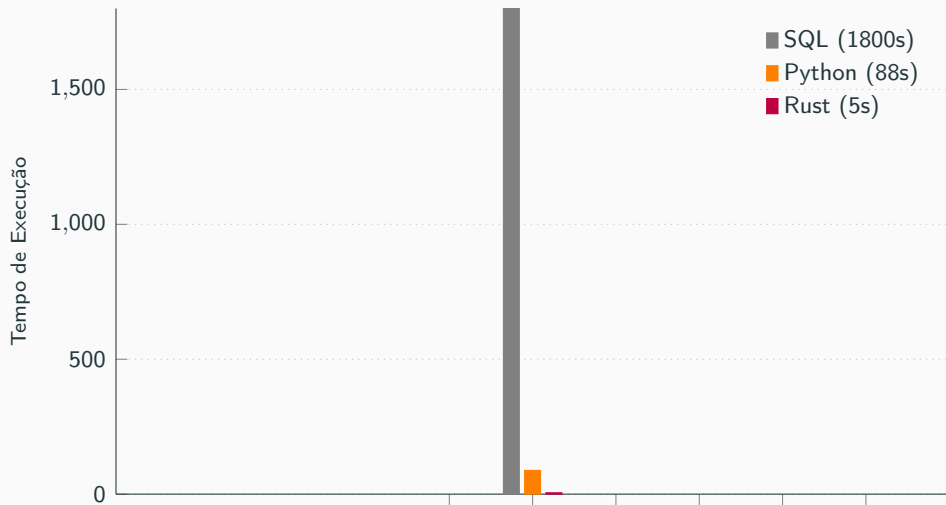
Benchmark (v2.0)



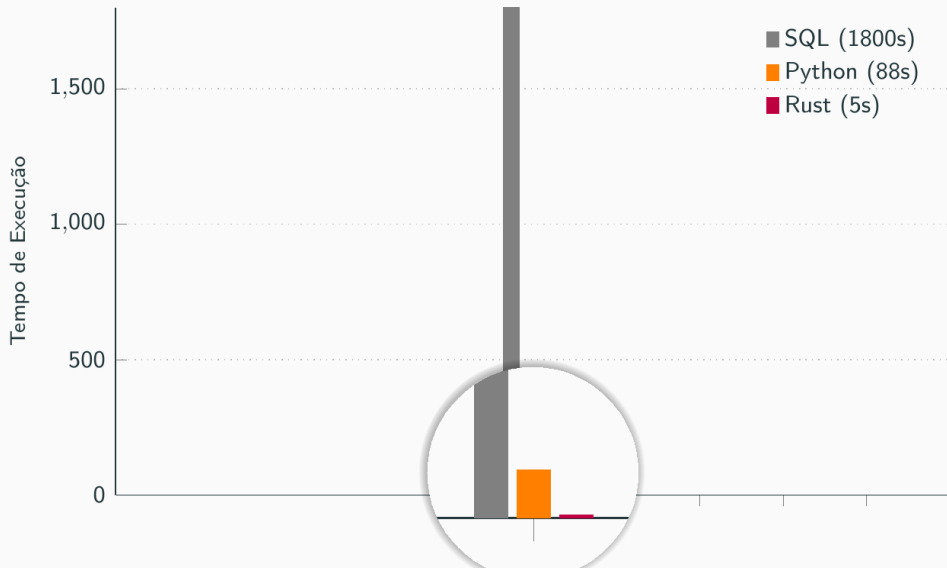
Benchmark (v2.0)



Benchmark (v2.0)



Benchmark (v2.0)





- Rust é uma linguagem recente, porém, completa em relação as ferramentas de desenvolvimento;
- Indicada para resolver problemas de processamento pesado (competindo com C, C++ e Fortran);
- Ótima para lidar com problemas que requerem paralelismo;
- Curva de aprendizado é grande;



Obrigado