The case for using Rust (as a marine engineer)

Nick Lamprinidis

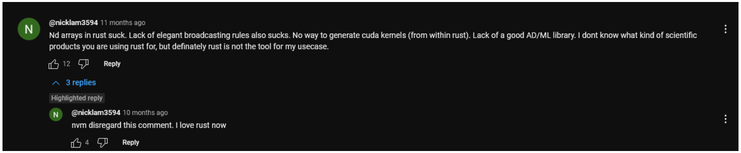
June 26, 2024

Outline

About me

- Working at ABS since 2018.
- Likes all things numerical.
- Not affiliated with the rust foundation.

Funny story


N
@nicklam3594 11 months ago
Nd arrays in rust suck. Lack of elegant broadcasting rules also sucks. No way to generate cuda kernels (from within rust). Lack of a good AD/ML library. I dont know what kind of scientific products you are using rust for, but definately rust is not the tool for my usecase.
12 Reply
3 replies
Highlighted reply
N
@nicklam3594 10 months ago
nvm disregard this comment. I love rust now
4 Reply

(youtube link: see last slide)

So what happened?

- Ported ABS's weather data processing library into rust.
- x10 times speed up vs julias NCDatasets library. `https://github.com/Alexander-Barth/NCDatasets.jl`
- Enter the rabit hole.

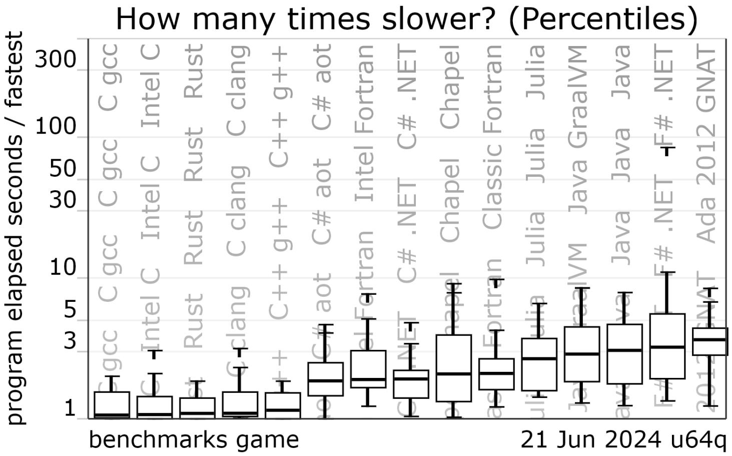
So what happened? (pt 2/2)

- Eventually ported everything into rust.
  - Weather hindcast/forecast analysis.
  - Orthodrome & pathfinding algorithms.
  - ML library.
  - Vessel Performance modeling.

What is rust?

- Initially developed by mozilla to solve the c++ problems in Firefox
- IT IS FAST
- Memory safety guaranteed
- cargo (package manager) is the best
- The best error error messages in any language
- RELIABLE (so much so it is boring)!
- Fearless concurrency.
- Great ecosystem + tooling.
- Open source.

How fast?


How many times slower? (Percentiles)
program elapsed seconds / fastest
300
100
50
30
10
5
3
1
C gcc
Intel C
Rust
C clang
C++ g++
C# aot
Intel Fortran
C# .NET
Chapel
Classic Fortran
Julia
Java GraalVM
Java
F# .NET
Ada 2012 GNAT
benchmarks game
21 Jun 2024 u64q

Safety

- Pointers checked at compile-time
- Thread-safety
- No hidden states
- Beautifull type system
- Error handling at its best

example 1

```
fn main() {
    let lista = vec![Food::Fasolakia(3), Food::Burger,
    let yummy_foods = yummy(&lista);
    println!("{:?}", yummy_foods);
}
fn yummy(lista: &Vec<Food>) -> Vec<bool> {// returns i
    let is_yummy = lista.iter().map(|food| match food
        Food::Fasolakia(_) => false,
        Food::Burger => true,
    });
    is_yummy.collect()
}
```

example 1 (Cont)

```
#[derive(PartialEq)]
enum Food {
    Fasolakia(u16),
    Burger,
    Gyros,
}
```

Run result

```
cargo run
   Compiling example1 v0.1.0 (C:\Users\NLamprinidis\Dow
error[E0004]: non-exhaustive patterns: ‘&Food::Gyros‘ n
  --> src/main.rs:9:50
   |
9  |     let is_yummy = lista.iter().map(|food| match f
   |                                                      ^
   |
note: ‘Food‘ defined here
  --> src/main.rs:18:6
   |
18 | enum Food {
   |      ^^^^
...
```

Run result (Cont)

```
21 |     Gyros,
   |     ----- not covered
   = note: the matched value is of type ‘&Food‘
help: ensure that all possible cases are being handled
   |
11 ~         Food::Burger => true,
12 ~         &Food::Gyros => todo!(),
   |

For more information about this error, try ‘rustc --ex
```

example 2

```
fn main() {
    let x = vec![1.0f32, 2.0, 3.0];
    let z = zero(x);
    let w = zero(x);
    println!("Hurray!!");
}

fn zero(x: Vec<f32>) -> Vec<f32> {
    x.iter().map(|a| a * 0.0).collect::<Vec<f32>>()
}
```

The error

```
error[E0382]: use of moved value: ‘x‘
 --> src/main.rs:5:18
  |
2 |     let x = vec![1.0f32, 2.0, 3.0];
  |         - move occurs because ‘x‘ has type ‘Vec<f32
4 |     let z = zero(x);
  |                  - value moved here
5 |     let w = zero(x);
  |                  ^ value used here after move
  |
```

The error (Cont)

```
note: consider changing this parameter type in functio
 --> src/main.rs:9:12
  |
9 | fn zero(x: Vec<f32>) -> Vec<f32> {
  |    ----    ^^^^^^^^ this parameter takes ownership
  |    |
  |    in this function
help: consider cloning the value if the performance co
  |
4 |     let z = zero(x.clone());
  |                   ++++++++
```

Final result

```
fn main() {
    let x = vec![1.0f32, 2.0, 3.0];
    let y = vec![4.0f32, 5.0, 6.0];
    let z = zero(&x);
    let w = zero(&x);
    println!("Hurray!!");
}

fn zero(x: &Vec<f32>) -> Vec<f32> {
    x.iter().map(|a| a * 0.0).collect::<Vec<f32>>()
}
```

example 3

```
// An integer division that doesn't ‘panic!‘
fn checked_division(dividend: i32, divisor: i32) -> Opt
    if divisor == 0 {
        // Failure is represented as the ‘None‘ variant
        None
    } else {
        // Result is wrapped in a ‘Some‘ variant
        Some(dividend / divisor)
    }
}
```

example 3 (Cont)

```
// This function handles a division that may not succee
fn try_division(dividend: i32, divisor: i32) {
    // ‘Option‘ values can be pattern matched, just lik
    match checked_division(dividend, divisor) {
        None => println!("{} / {} failed!", dividend, d
        Some(quotient) => {
            println!("{} / {} = {}", dividend, divisor,
        },
    }
}
```

example 3 (Cont)

```
fn main() {
    let x = checked_division(4, 2);
    let y = checked_division(1, 0);
    // Unwrapping a `Some` variant will extract the va
    println!("x is {:?}", x.unwrap());
    // proper error handling
    match y {
        Some(v) => println!("y is {:?}", v),
        None => println!("y is None"),
    }
    // Unwrapping a `None` variant will `panic!`
    println!("y is {:?}", y.unwrap());
}
```

Cons

- Slower dev time (debatable).
- Very slow compile times.
- Syntax is verbose (kind of).
- Steep learning curve.

Some general rule-of-thumbs

- Only use Vec<stuff> and structs to store data
- Functions on the above should accept &Vec<stuff> and &Struct
- Ignore Generics and Traits for now
- Dont mind .clone()

Numerical Example

min $L = (x + y)^2$
given that $x^2 + y^2 = 1$
Rewritting this as lagrange multipliers
min $L = (x + y)^2 + \lambda * (x^2 + y^2)$
Rewritting this as lagrange multipliers (heuristic)
min $L = (x + y)^2 + \lambda * (x^2 + y^2)^2$

Questions?

Repo & contact info

- github repo:
  `https://github.com/krestomantsi/opada-2024`
- email: nlampri@gmail.com
- (youtube link: `https://www.youtube.com/watch?v=0JkbNFpXlXc&lc=UgwQJyFb6m1vBkg431d4AaABAg.9sIktyoda_P9t3lAdkUZLB`)