

Why Is Rust the Rising Star?

David Chocholatý

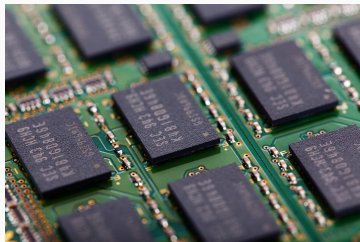
June 14, 2024

Brno University of Technology

Why Rust Matters?

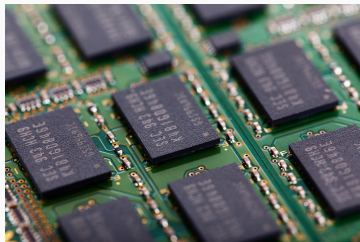
Why Rust Matters?

- Memory Safe Programming Language



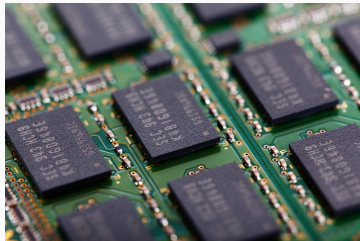
Why Rust Matters?

- Memory Safe Programming Language
- Performance Comparable to C and C++



Why Rust Matters?

- Memory Safe Programming Language
- Performance Comparable to C and C++
- Low-Level Control

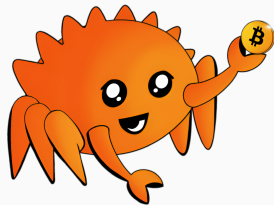


Stay Safe!



Where Can I Use Rust?

- Systems Programming & CLI Tools
- Concurrency and Parallelism
- Network Programming
- Embedded Systems
- Game Development
- High-Performance Computing
- Blockchain and Cryptography



- **National Security Agency (USA) urges shift to memory safe programming languages**
 - Published: Nov 12, 2022
- **Why Rust is one of the world's most cherished programming languages**
 - Published: Feb 26, 2024
- **White House urges developers to dump C and C++**
 - Published: Feb 27, 2024
- **Why Rust is emerging as developers' favourite programming language**
 - Published: May 28, 2024

We Want the Safety



We Want the Safety



What Is Rust?

What Is Rust?

- Safety, Speed, and Concurrency
- Developed by Mozilla (Graydon Hoare)
- First Stable Release in 2015
- Now an Open-Source Project with an Active Community
- Extensive Documentation, Tutorials, and Tools

What Do We Mean by **Safe**?

What Do We Mean by **Safe**?

Does It Mean That Rust Is **100%** Memory Safe?

What Is Rust?

What Do We Mean by **Safe**?

Does It Mean That Rust Is **100%** Memory Safe?

What Is the Reality of **Performance**?

Memory Safety

Memory Safety — C++ Example

C++

```
void addBeer(std::vector<int>& beers) {  
    beers.push_back(0); // Potential buffer overflow if vector was manually managed!  
}  
  
int main() {  
    int* beerFridge = new int[5]; // Yeah, new fridge!  
    delete[] beerFridge; // Throw away the scrap!  
    std::cout << beerFridge[0]; // Use-after-free.  
  
    std::vector<int> beerCollection(5);  
    addBeer(beerCollection); // Add another beer.  
  
    int* lostBeer = new int(42); // A lovely special beer.  
    // Forgot to delete lostBeer, potential memory leak here!  
  
    return 0; // Potential memory leak due to lostBeer!  
}
```

Memory Safety — Rust Example

Rust

```
fn add_beer(beers: &mut Vec<i32>) {  
    beers.push(0); // Safe, Rust's Vec handles resizing.  
}  
  
fn main() {  
    let beer_fridge = vec![0; 5]; // A new fridge.  
    drop(beer_fridge); // Throw it away!  
  
    // println!("{}", beer_fridge[0]); // Leads to compile-time error.  
  
    let mut beer_collection = vec![0; 5];  
    add_beer(&mut beer_collection); // Rust's borrowing.  
  
    let lost_beer = Box::new(42); // Special beer.  
    // Automatically deallocated when it goes out of scope.  
  
    // No manual management needed.  
}
```

Type System

C++

```
// Base class representing a dessert.  
class Dessert {  
public:  
    virtual ~Dessert() = default; // Destructor allowing polymorphic behavior and memory  
        management.  
    virtual double yumminess() const = 0; // Pure virtual function, making Dessert an abstract  
        base class.  
};
```

Type System — C++ Example

C++

```
int main() {  
    // The use of unique pointers (std::unique_ptr).  
    std::vector<std::unique_ptr<Dessert>> desserts;  
    desserts.push_back(std::make_unique<Cake>(5.0)); // Sweet cake!  
    desserts.push_back(std::make_unique<Cookie>(4.0, 6.0)); // Crunchy cookie with lots of  
        choco chips!  
  
    // ...  
  
    // Accessing memory after deallocation.  
    Cake* danglingPointer = dynamic_cast<Cake*>(desserts[0].get());  
    desserts.clear(); // Deallocate memory.  
  
    // Attempt to access memory through the dangling pointer.  
    danglingPointer->candleCount(); // Implemented only for cakes.  
  
    return 0;  
}
```

Rust

```
// Define an enum representing different types of desserts.  
enum Dessert {  
    // Cake dessert with sweetness as a parameter.  
    Cake { sweetness: f64 },  
    // Cookie dessert with crunchiness and choco_chips as parameters.  
    Cookie { crunchiness: f64, choco_chips: f64 },  
}
```

Type System — Rust Example

Rust

```
impl Dessert {  
    // Method to calculate the yumminess of the dessert.  
    fn yumminess(&self) -> f64 {  
        match self {  
            // Match on the enum variant to calculate yumminess based on dessert type.  
            Dessert::Cake { sweetness } => *sweetness * 1.5,  
            Dessert::Cookie { crunchiness, choco_chips } => crunchiness * 2.0 + choco_chips *  
                3.0,  
        }  
    }  
}
```

Type System — Rust Example

Rust

```
// Create a vector of desserts containing Cake and Cookie instances.
fn main() {
    let desserts = vec![
        Dessert::Cake { sweetness: 5.0 },
        Dessert::Cookie { crunchiness: 4.0, choco_chips: 6.0 },
    ];

    // ...

    // Rust's ownership system prevents access to dangling pointers.
}
```


Fixed!



Unsafe Rust

Unsafe Rust — C++ Example

C++

```
int main() {  
    int donuts[] = {1, 2, 3, 4, 5}; // Mmm... donuts!  
    int* bart = donuts; // Bart's pointer to the donuts.  
  
    std::cout << "Ay_caramba!" << std::endl;  
  
    for (int i = 0; i < 5; ++i) { // Bart's prank - doubling the donuts!  
        // No bounds checking.  
        *(bart + i) = *(bart + i) * 2; // Unsafe pointer arithmetic.  
    }  
  
    std::cout << "Uh_oh!_Bart's_modified_donuts:_";  
    for (int i = 0; i < 5; ++i) {  
        std::cout << donuts[i] << "_";  
    }  
    std::cout << "Woo-hoo!" << std::endl;  
  
    return 0;  
}
```

Unsafe Rust — Rust Example

Rust

```
fn main() {  
    let mut donuts = [1, 2, 3, 4, 5];  
    let bart = donuts.as_mut_ptr(); // Bart's pointer to the donuts.  
  
    println!("Ay_caramba!");  
  
    unsafe {  
        for i in 0..5 { // Bart's prank - doubling the donuts!  
            // No bounds checking.  
            *bart.add(i) *= 2; // Unsafe pointer arithmetic.  
        }  
    }  
  
    print!("Uh_oh!_Bart's_modified_donuts:");  
    for i in 0..5 {  
        print!("{}", donuts[i]);  
    }  
    println!("Woo-hoo!");  
}
```

Performance

Performance

- The Benchmarks Game (<https://bit.ly/3X4NLGI>)

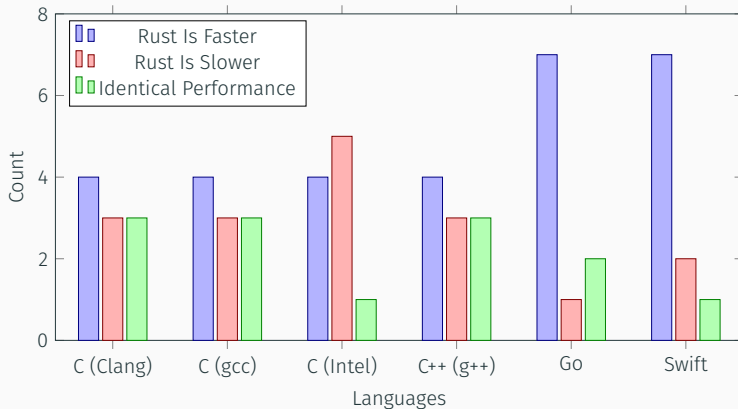


Figure 1: Comparison of Rust Performance with Other Languages

Presentation Hosted on GitHub

- The Presentation and Source Codes Available on:

<https://bit.ly/3KsMh1y>



Coding in Rust Is Hard ...



But It Makes Sense!



Why Is Rust the Rising Star?

Questions?

References i



crates.io.

`https://crates.io/`.

Accessed: 2024-05-30.



T. Act.

Why rust is one of the world's most cherished programming languages.

`https://www.thirdact.se/en/news/`

`why-rust-is-one-of-the-worlds-most-cherished-programming-languages`,
2024.

Accessed: 2024-05-30.



N. S. Agency.

U.s. and international partners issue recommendations to secure software products.

*[https://www.nsa.gov/Press-Room/Press-Releases-Statements/
Press-Release-View/Article/3608324/
us-and-international-partners-issue-recommendations-to-secure-software-products](https://www.nsa.gov/Press-Room/Press-Releases-Statements/Press-Release-View/Article/3608324/us-and-international-partners-issue-recommendations-to-secure-software-products)*

2023.

Accessed: 2024-05-24.



AhmadTurk.

Minion.

<https://www.deviantart.com/ahmadturk/art/Minion-428929017>, 2014.

Accessed: 2024-05-28.



BairesDev.

When speed matters: Comparing rust and c++.

<https://www.bairesdev.com/blog/when-speed-matters-comparing-rust-and-c/>, 2024.

Accessed: 2024-05-25.



T. R. Contributors.

Rustlings.

<https://github.com/rust-lang/rustlings>.

Accessed: 2024-05-29.






T. R. P. L. Contributors.

The rust programming language book.

<https://github.com/rust-lang/book>.



Accessed: 2024-05-29.



References iv



-  M. Corporation.
Rust mascot ferris bitcoin transparent.
https://commons.wikimedia.org/wiki/File:Rust_Mascot_Ferris_Bitcoin_Transparent.png, 2024.
Accessed: 2024-05-29.
-  T. Down.
Nsa urges shift to memory safe programming languages.
<https://www.threatdown.com/blog/nsa-urges-shift-to-memory-safe-programming-languages/>, 2024.
Accessed: 2024-05-24.
-  P. film s.r.o.
Pat mat.
https://commons.wikimedia.org/wiki/File:Pat_Mat.jpg, 2014.
Accessed: 2024-06-02.

References v

-  G. Gross.
White house urges developers to dump c and c++.
<https://www.infoworld.com/article/3713203/white-house-urges-developers-to-dump-c-and-c.html>, 2024.
Accessed: 2024-05-23.
-  imgur.
Gotta polish the all seeing orb.
<https://imgur.com/gotta-polish-all-seeing-orb-BVorcID?r>, 2016.
Accessed: 2024-06-02.

-  R. Mey.
Airbus plane departure lufthansa.
https://pixabay.com/photos/airbus-plane-departure-lufthansa-8607152/,
2024.
Accessed: 2024-05-27.
-  U. D. of Defense.
Software memory safety.
Technical report, 2022.
Accessed: 2024-05-23.

-  PublicDomainPictures.
Board, card, chip.
https://pixabay.com/photos/board-card-chip-computer-data-22098/,
Unknown.
Accessed: 2024-05-29.
-  M. Safety.
Memory safety documentation.
https://www.memorysafety.org/docs/memory-safety/, 2024.
Accessed: 2024-05-24.

-  Skoolcool.
This is fine too.
https://www.deviantart.com/skoolcool/art/This-is-fine-too-678378168,
2017.
Accessed: 2024-05-28.
-  B. C. Software.
Memory safety: What's the big deal? part i.
https://www.linkedin.com/pulse/memory-safety-whats-big-deal-part-i-buildable-pv1ic, 2023.
Accessed: 2024-05-24.

References ix



B. G. Team.

The computer language benchmarks game: Rust programs.

<https://www.bairesdev.com/blog/when-speed-matters-comparing-rust-and-c/>, 2024.

Accessed: 2024-05-24.



T. R. D. Team.

Rust by example.

<https://doc.rust-lang.org/rust-by-example/>.

Accessed: 2024-05-29.



The Awesome Rust Contributors.

Awesome rust.

<https://github.com/rust-unofficial/awesome-rust>.

Accessed: 2024-05-29.

References x



T. N. Web.

Why rust developers' favourite programming language.

<https://thenextweb.com/news/>

[why-rust-developers-favourite-programming-language](#), 2024.

Accessed: 2024-06-01.



Wikilimages.

Rocket launch.

[https:](https://pixabay.com/photos/rocket-launch-rocket-lift-up-nasa-67721/)

[//pixabay.com/photos/rocket-launch-rocket-lift-up-nasa-67721/](#),

2024.

Accessed: 2024-05-27.