HPS

RUST Programming for HPC Application

Yuvraj Singh

Table of contents

RUST History

- Invented by - Graydon Hoare
- Development Focus
  - Network Services
  - Command Line Applications
  - WebAssembly (WASM)
  - Embedded Devices
- 4 Epochs
  - Personal years (2006 - 2010)
  - The Graydon years (2010 - 2012)
  - The Type-system years (2012 - 2014)
  - The Release year (2015 - 2016)
- First Release - RUST 1.0
- Current Release - RUST 1.62.0 (30 June, 2022)

RUST Features

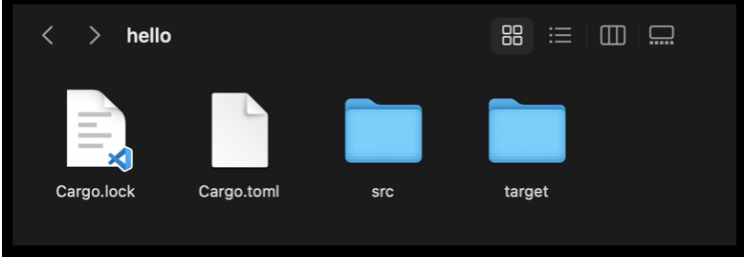
- Systems Programming Language
- Statically Typed Language
- Memory Safety
- Thread Safety
- Efficient C Bindings
- 2 MODES
  - Safe RUST (Default)
  - Unsafe RUST

RUST Development Tools

- Compiler
  - rustc
- Package Manager
  - cargo
- Editor Support Examples
  - Vim, Emacs, Kate and gedit
- IDE Support Examples
  - VScode and Eclipse
- Version Translator
  - RustFix

RUST "Hello, World!"

```
(base) yuvrajsingh@Yuvrajs-MBP ~ % cargo new hello
     Created binary (application) `hello` package
(base) yuvrajsingh@Yuvrajs-MBP ~ % cd hello
(base) yuvrajsingh@Yuvrajs-MBP hello % cargo build
   Compiling hello v0.1.0 (/Users/yuvrajsingh/hello)
    Finished dev [unoptimized + debuginfo] target(s) in 3.77s
(base) yuvrajsingh@Yuvrajs-MBP hello % cargo run
    Finished dev [unoptimized + debuginfo] target(s) in 0.00s
     Running `target/debug/hello`
Hello, world!
(base) yuvrajsingh@Yuvrajs-MBP hello %
```



RUST "Hello, World!"

main.rs

```rust
fn main() {
    println!("Hello, world!");
}
```

Cargo.toml

```toml
[package]
name = "hello"
version = "0.1.0"
authors = ["yuvrajsingh"]
edition = "2018"

# See more keys and their definitions at https://doc.rust-lang.org/cargo/reference/manifest.html

[dependencies]
log = "0.4"
ndarray = "0.10.0"
num = "0.2"
ocl = "0.16.0"
```

Data-Types (1)

Integer Types

Length	Signed	Unsigned
8-bit	`i8`	`u8`
16-bit	`i16`	`u16`
32-bit	`i32`	`u32`
64-bit	`i64`	`u64`
128-bit	`i128`	`u128`
arch	`isize`	`usize`

Integer Literals

Number literals	Example
Decimal	`98_222`
Hex	`0xff`
Octal	`0o77`
Binary	`0b1111_0000`
Byte (`u8` only)	`b'A'`

Data-Types (2)

- Float Type
  - f32 and f64
- Boolean Type
  - true and false
- String Type
  - str and String
- Also supports
  - Arrays
  - Tuples
  - Structs and Enums

RUST Variables and Mutability

- Variables
  - Declared using "let" keyword
  - "mut" keqord makes variable mutable E.g.-> let mut i8
- Constants
  - Declared using "const"
  - Immutable
- Shadowing

RUST's Ownership Feature

- Responsible for memory-safety and high performance
- Prevents Bugs
- Manages Memory
- Rules
  - each variable has owner
  - one owner at a time
  - value drops when out of scope

RUST's Borrowing Feature

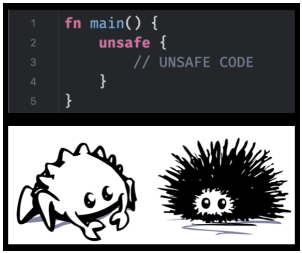
- Shares variable's past
  - uses reference
- Temporary owner
- Cannot exceed scope of owner
- Immutable by default
  - ONLY one mutation possible

RUST's Fearless Concurrency Feature

- Goal
  - Handling concurrent programming safely and efficiently
- Based on
  - Ownership
  - Type System

Unsafe RUST

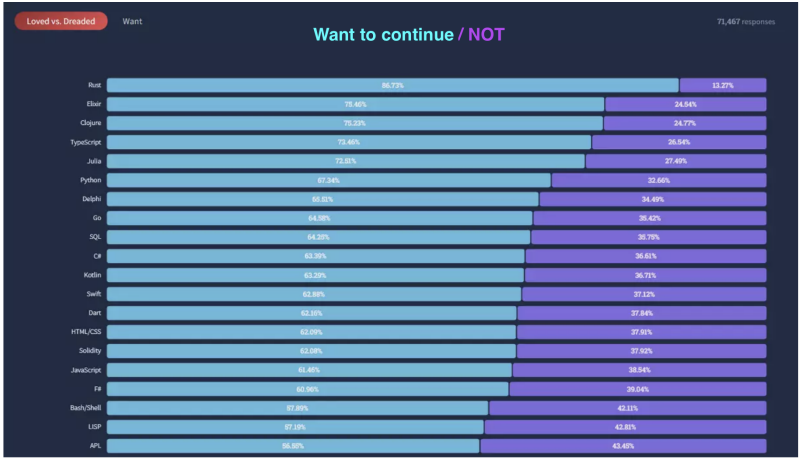
- "unsafe" keyword
- Unsafe Superpowers
  - Calling unsafe functions
  - Dereference a raw pointer
  - Implement an unsafe trait
  - Access or modify a mutable static variable
- Unsafe Disadvantage
  - Memory Unsafety



RUST's Popularity

[3] Interest in Rust programming language over time
(May 2012-May 2022, generated from google trends)

RUST's Popularity



2022 Stack Overflow Developer Survey: The 20 most popular programming languages
(Image: Stack Overflow)

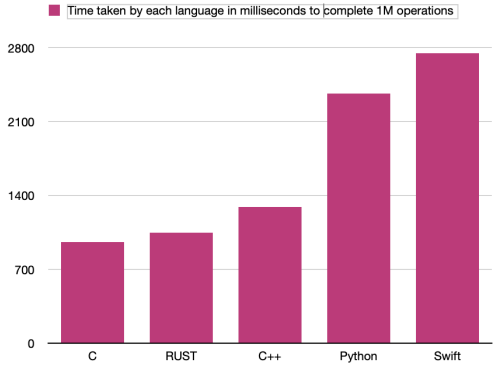
Outline

General Comparision

[1] Overview of different possible programming languages

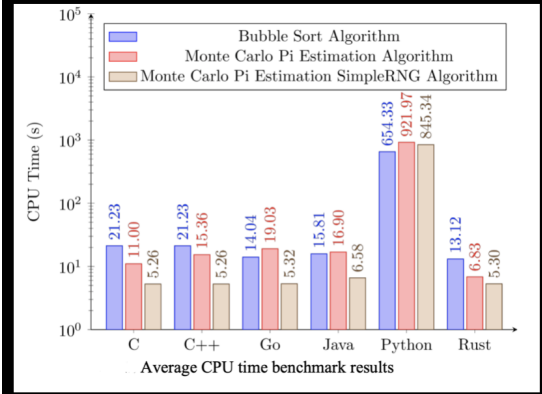
Language	Aspect	Memory management	Systems program.	Bare-metal
Rust	Zero-cost safety	Automatic	Yes	Yes
C	Currently in use	Manual + Automatic	Yes	Yes
Ada	Safety critical apps	Manual + Automatic	Yes	Yes
Python	Interpreted	Garbage Collected	No	No
Java	JIT Compiler	Garbage Collected	No	No
Go	Modern language	Garbage Collected	Yes	Not officially
Haskell	Functional	Garbage Collected	No	No

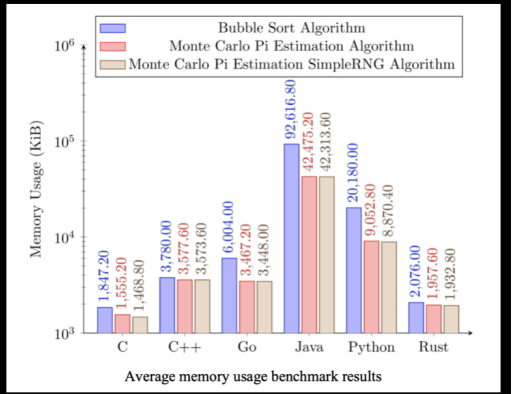
Performance Comparison




```rust
fn main() {
let mut i = 0;
while i < 1000000
{
    println!("hello");
    i = i + 1;
}
}
```

Performance and Memory Usage Comparison





[3]

Outline

1. RUST-ArrayFire, a library for parallel computing

- Library for parallel computing with an API
- RUST-ArrayFire mechanism
  - ArrayFire abstraction
  - Memory manager
- Array, as generic container type
- Additional supported datatypes
  - C32 -> complex single-precision
  - C64 -> complex double-precision
  - B8 -> 8-bit boolean values
  - F16 -> 16-bit float number
- Setup
  - Min. RUST version "1.31"
  - DEPENDENCIES -> arrayfire = "*"

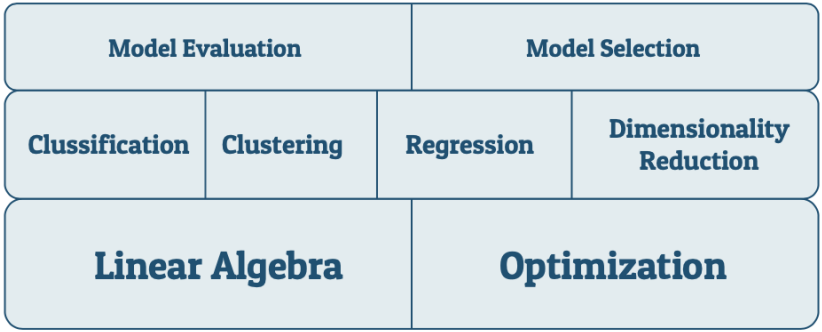
2. RUSTA-CUDA, an interface to NVIDIA CUDA Driver API

- RUST-CUDA design
  - High-level
  - Safe and fast
- Setup
  - CUDA Version 8.0 or newer
  - CUDA Capable GPU
  - DEPENDENCIES -> rustacuda = "*"

3. RUST-SmartCore, library for Machine learning

- Features
  - Support most ML algorithms
  - NO hard dependencies
- Include tools for
  - Linear-algebra
  - Optimization
  - Scientific-computing
- Setup
  - DEPENDENCIES -> smartcore = "*"
  - "ndarray" and "nalgebra"

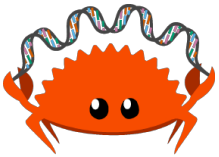
RUST-SmartCore, library for Machine learning



[7] SmartCore's architecture represented as layers.

4. RUST-BIO, a library for bioinformatics

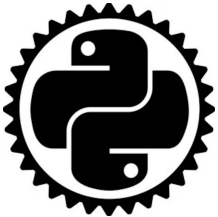
- RUST-BIO design
  - Bioinformatic algorithms
  - Bioinformatic data structures (e.g. alphabets)
- Some tools for Bioinformatics
  - Major pattern matching algorithms
  - Convenient alphabet implementation
  - Pairwise alignment
  - q-gram index
- Setup
  - Min. RUST version "1.53.0"
  - DEPENDENCIES -> bio = " * "
- Performance equivalent to C++



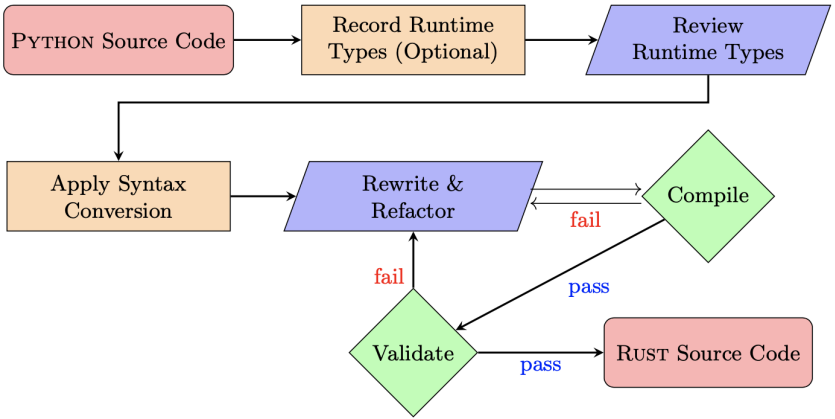
Outline

Transpiling Python to Rust

- Objective
  - High performance implementations
- Advantages
  - Semiautomatic procedure
  - Retains readability
  - Easy to switch
- Tools required
  - pyrs
  - MonkeyType
  - IntelliJ



Transpiling Python to Rust



[4] Transpiling Python to Rust.

Transpiling Python to Rust for Black-Scholes Model

- Model for Dynamics of financial markets
- Results based on
  - Time consumed
  - Memory consumed
  - Efforts consumed
- Result

[4] Execution profile of Black–Scholes on PC (allocations included)

Black–Scholes	**Python (`MKL`)**	**Rust (native)**
Execution time	27.29 s	11.70 s
Peak memory consumption	9.372 GB	3.456 GB

RUST for astrophysics

- Advantages
  - Fast
  - Provides Safety
  - More Accurate results
- Example - N-Body Dynamical simulator
  - Compared with Fortran, C and Go

RUST for astrophysics Benchmarks

- Machine Used
  - 1,6 GHz Intel Core i5
- Time taken to calculate positions for the two particles after 1 million years.
- Result

Best execution times of pure N-body simulations, for an integration time of 1 million years using a leap-frog integrator. **[5]**

Rust	Fortran	C	Go
0m13.660s	0m14.640s	2m32.910s[a]	4m26.240s

Outline

Conclusion

- RUST is a GOOD candidate to performance HPC.
- RUST is getting better (inc. HPC Libraries, ecosystem, etc.).
- RUST has variety of HPC Libraries.
- RUST is very well documented.
- RUST is safe a language.

Sources

1. Nico Borgsmüller, RUST for Embedded Software Development
2. Michal Sudwoj, RUST in HPC environment
3. William Bugden, RUST for Safety and Performance
4. Timo Hämäläinen, Transpiling Python to RUST
5. Sergi Blanco-Cauresma, RUST for AstroPhysics ?
6. www.rust.org
7. https://smartcorelib.org