JULY 28, 2026

A Performance and Usability Study of the Rust Programming Language

Riley Fisher
SIParCS Graduate Intern

Mentors: Lalo Torres, Ren Stengel, & John Dennis

This material is based upon work supported by the NSF National Center for Atmospheric Research, which is a major facility sponsored by the U.S. National Science Foundation under Cooperative Agreement No. 1852977. Any opinions, findings and conclusions or recommendations expressed in this material do not necessarily reflect the views of NSF.

Motivation

Current state

NCAR models contain code that is unable to utilize modern computing tools.

+

Better practices

Prioritize developability and usability when choosing a programming language.

+

Maintain performance

Match or exceed the performance of current implementations.

Which programming languages are well-suited for NCAR's community models?

Explanation of Problem

Goal: benchmark the performance and usability of a simple fluid model between programming languages on Derecho

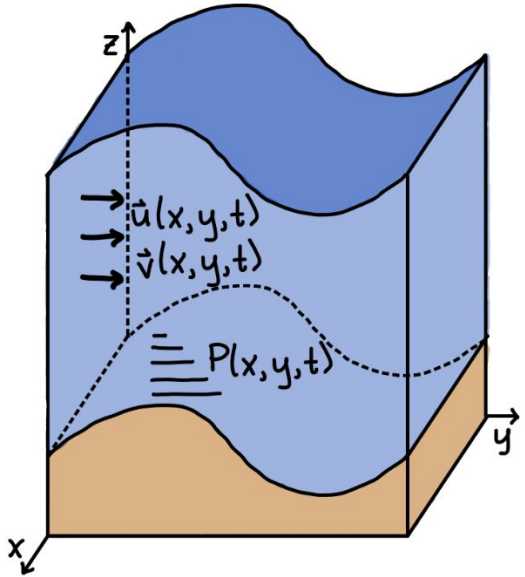
Explanation of Problem

Goal: benchmark the performance and usability of a simple fluid model between programming languages on Derecho

Choice of Model: 2D Shallow Water Equations

- Models velocity and pressure of an incompressible fluid
- Fluid is “shallow”, i.e. the vertical scale is much smaller than the horizontal scale
- Examples: propagation of a tsunami, storm surge

Numerical method: time-filtered leapfrog scheme with second-order central finite differences



Explanation of Problem

Goal: benchmark the performance and usability of a simple fluid model between programming languages on Derecho

Programming Languages

- Rust
- C
- Python
- Fortran

Explanation of Problem

Goal: benchmark the performance and usability of a simple fluid model between programming languages on Derecho

Programming Languages

- Rust
- C
- Python
- Fortran

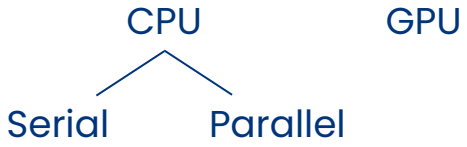
Explanation of Problem

Goal: benchmark the performance and usability of a simple fluid model between programming languages on Derecho

Programming Languages

- **Rust**
- **C**
- **Python**
- **Fortran**

Implementations



Explanation of Problem

Goal: benchmark the performance and usability of a simple fluid model between programming languages on Derecho

Programming Languages

- **Rust**
- **C**
- **Python**
- **Fortran**

Implementations

CPU GPU

Serial Parallel

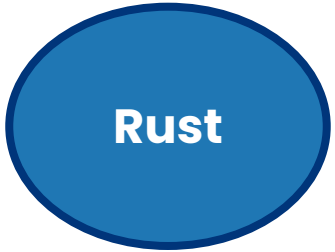
Performance Criteria

- Runtime
- Strong Scaling
- **Usability**

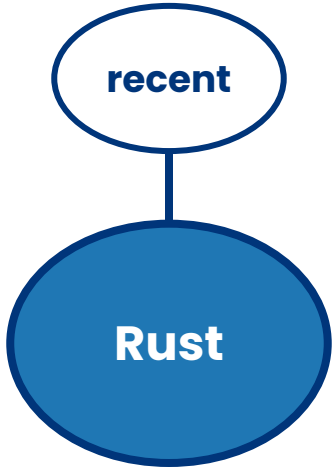
*Icons made by Freepik from www.flaticon.com

What is Rust?

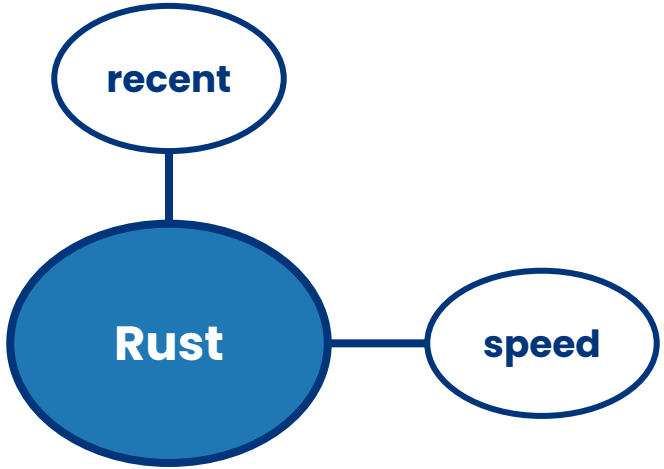
NCAR
OPERATED BY UCAR



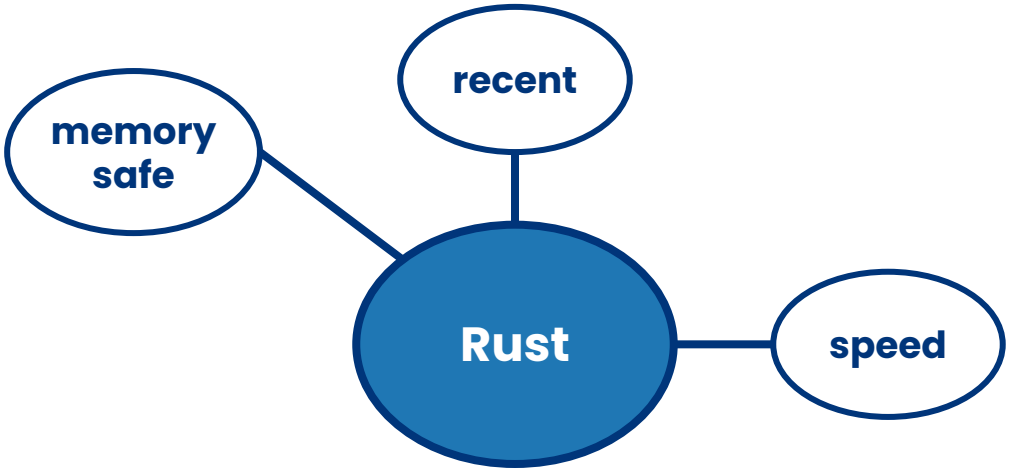
What is Rust?



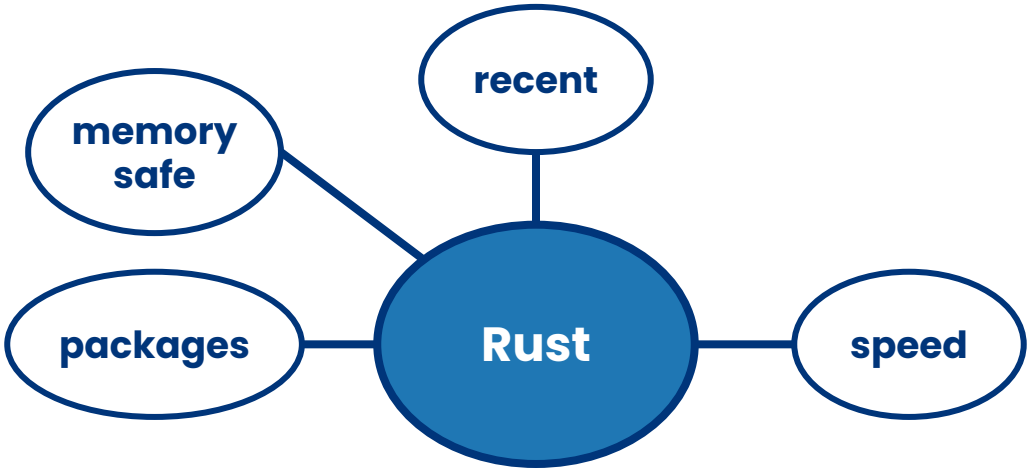
What is Rust?



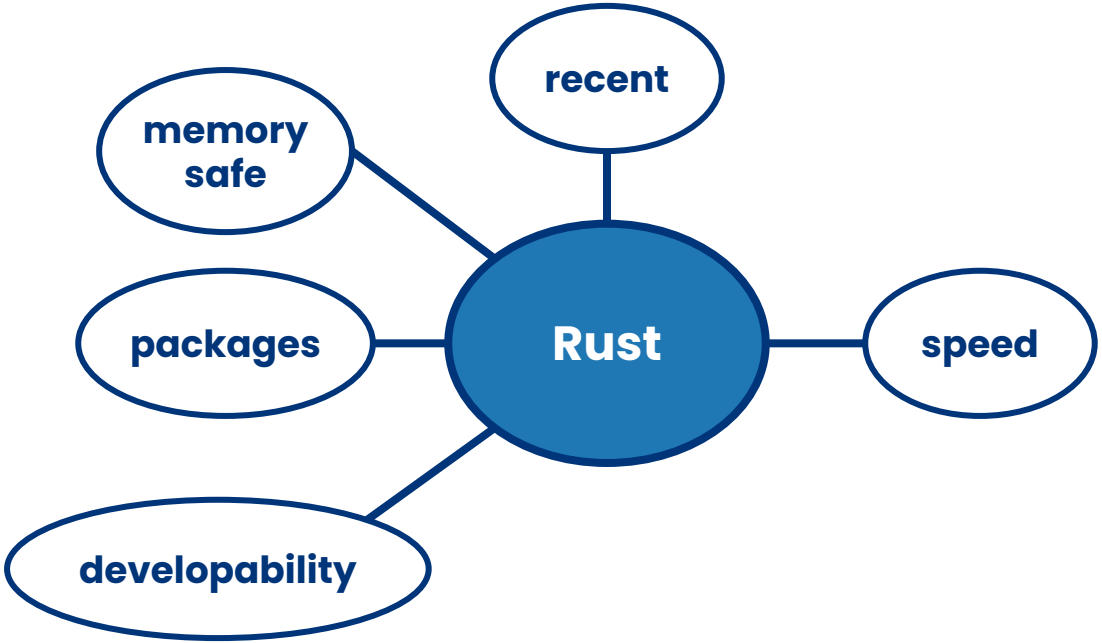
What is Rust?



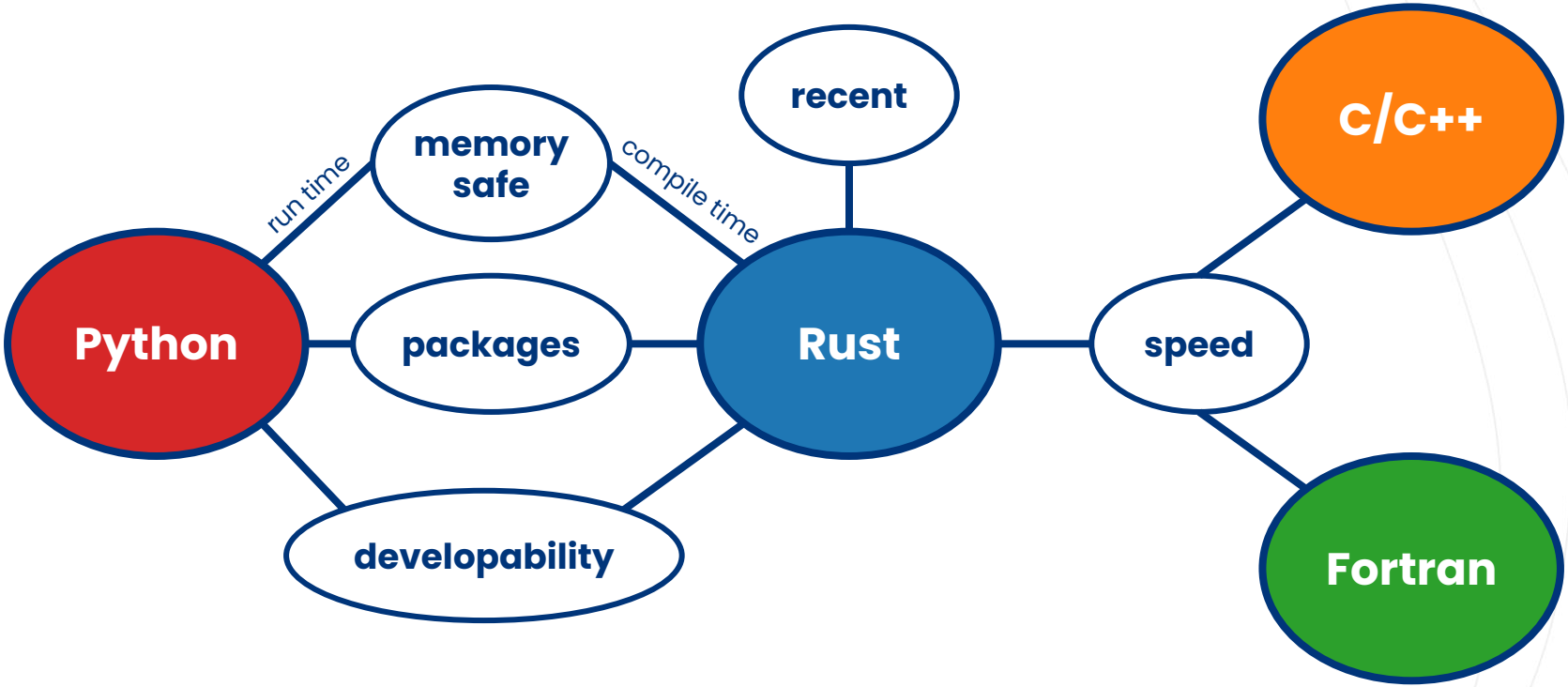
What is Rust?



What is Rust?



What is Rust?


recent
memory safe
run time
compile time
C/C++
Python
packages
Rust
speed
developability
Fortran

What is Rust?

NCAR
OPERATED BY UCAR

C/C++

Python

Rust

Fortran

What is Rust?

NCAR
OPERATED BY UCAR

Python

Rust

C/C++

```
$ gcc project.c -o project
$ ./project
```

Fortran

```
$ gfortran project.f90 -o project
$ ./project
```

What is Rust?

Python

Rust

C/C++

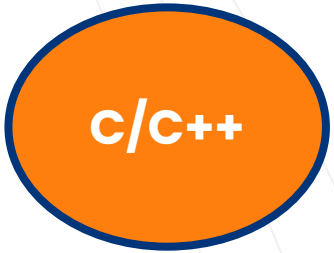
```
$ gcc project.c -o project
$ ./project
```

```
$ cargo run
   Compiling project v0.1.0 ($USER/project-dir)
    Finished `dev` profile target(s) in 0.10 secs
     Running `target/debug/project`
```

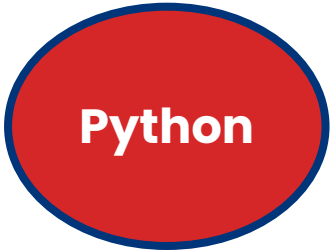
Fortran

```
$ gfortran project.f90 -o project
$ ./project
```

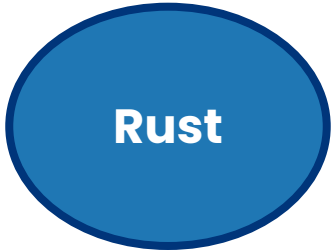
What is Rust?




```
$ gcc project.c -o project
$ ./project
```




```
$ python project.py
```

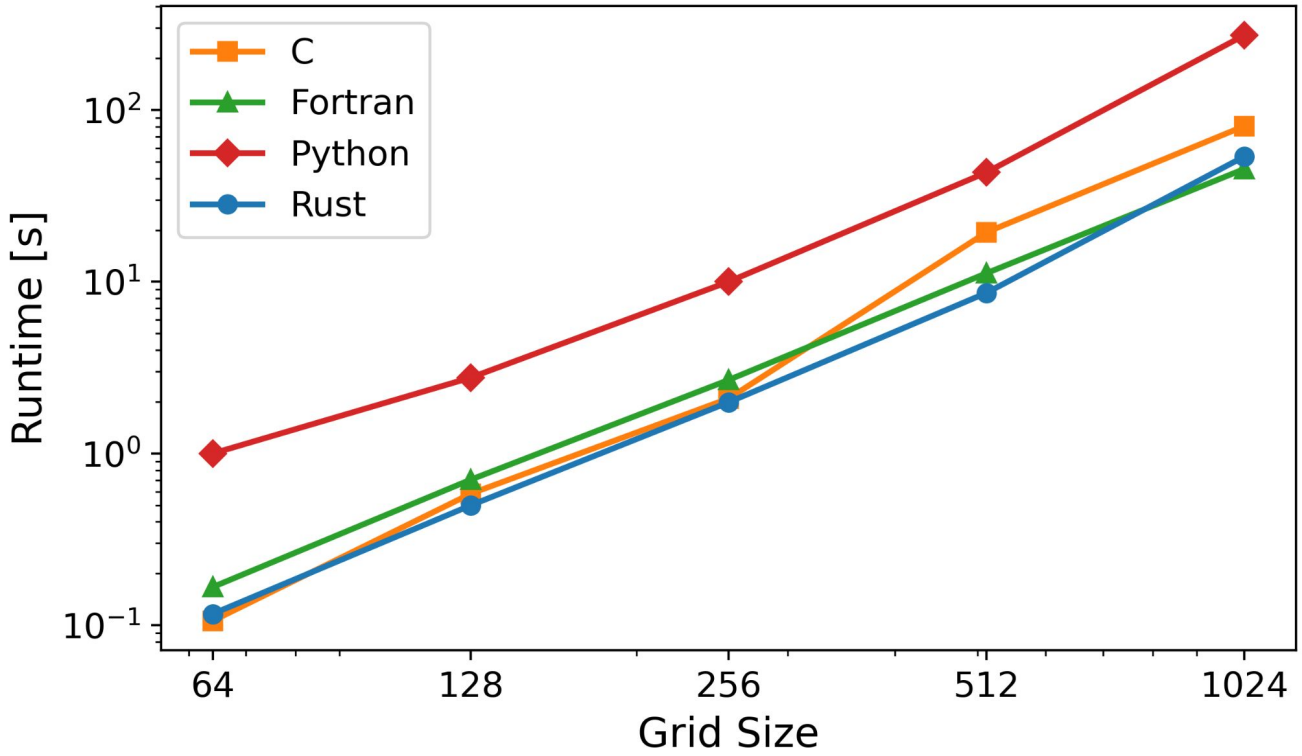



```
$ cargo run
   Compiling project v0.1.0 ($USER/project-dir)
    Finished `dev` profile target(s) in 0.10 secs
     Running `target/debug/project`
```

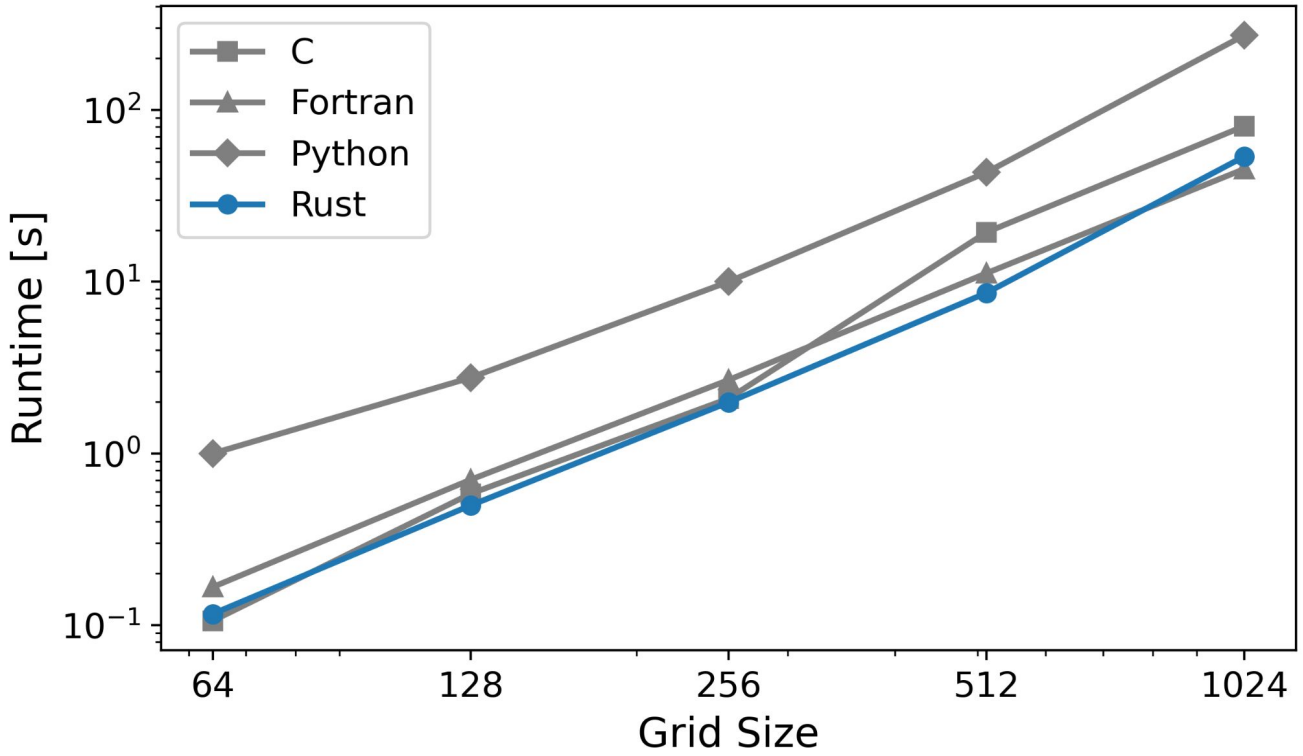

```
$ gfortran project.f90 -o project
$ ./project
```

Performance Results

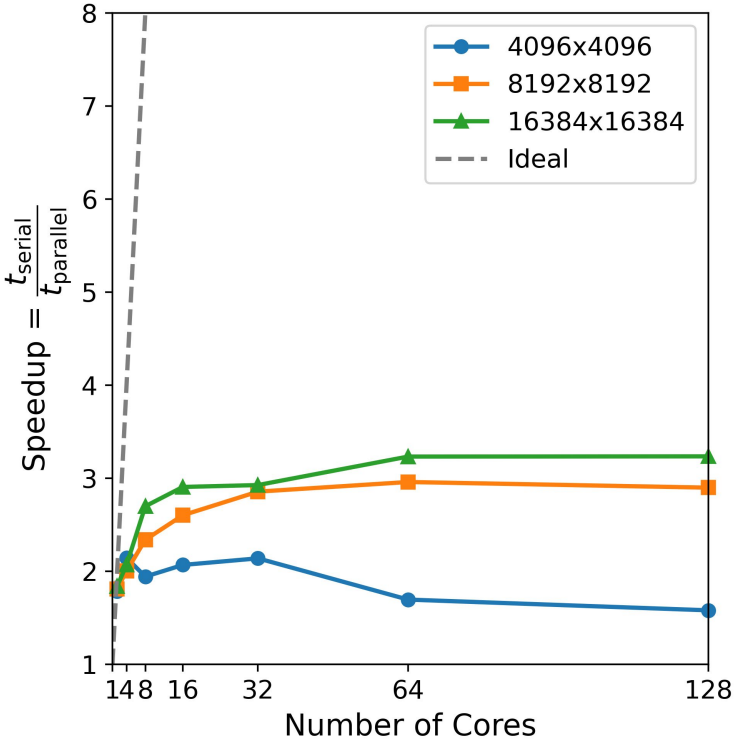
Serial, Single Core CPU

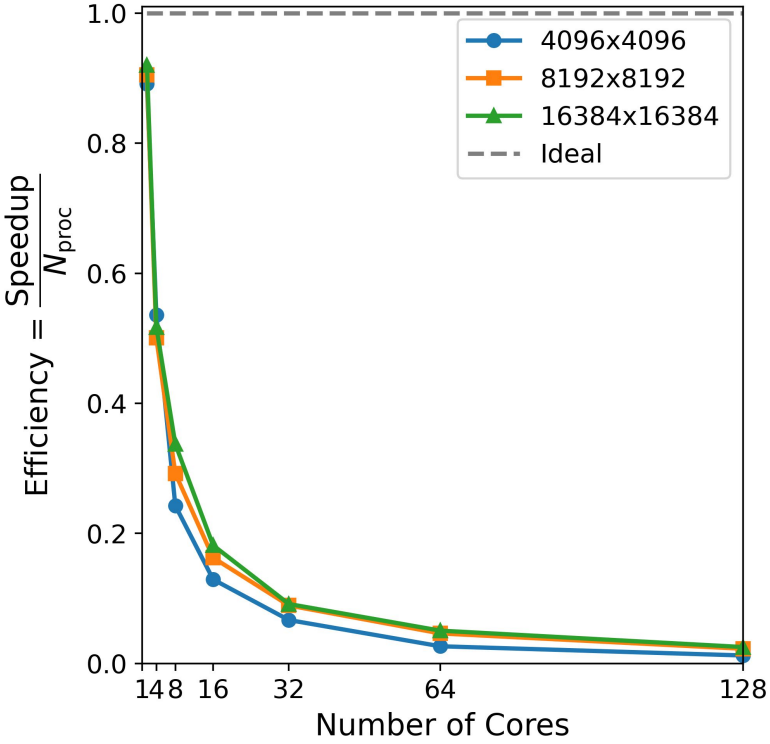


Serial, Single Core CPU

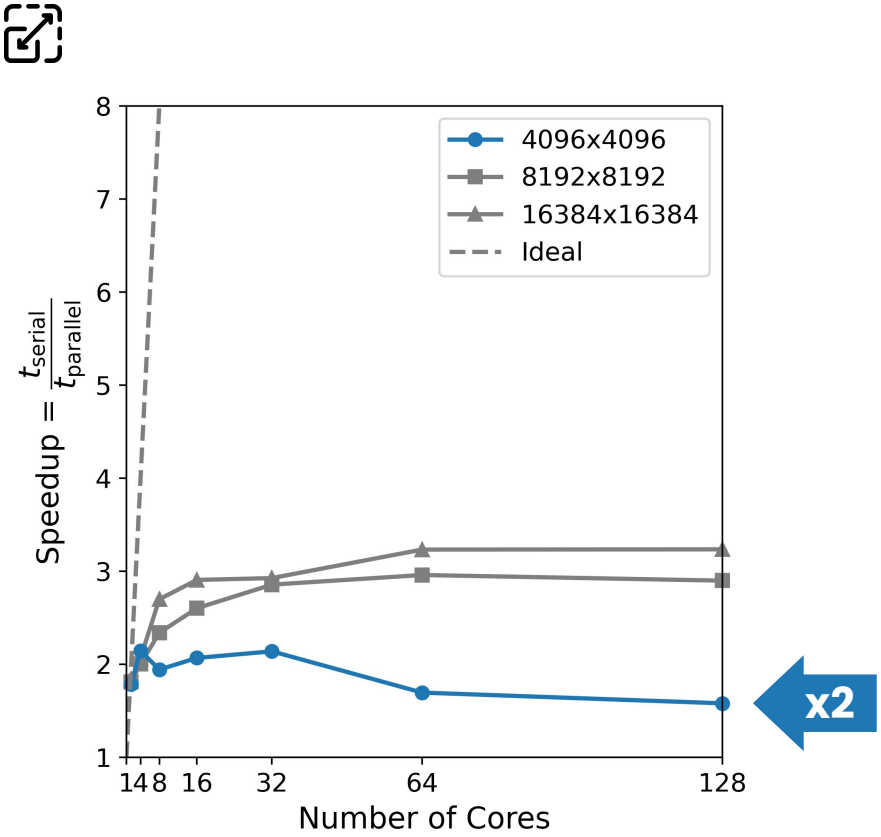


Shared Memory, Single Node CPU

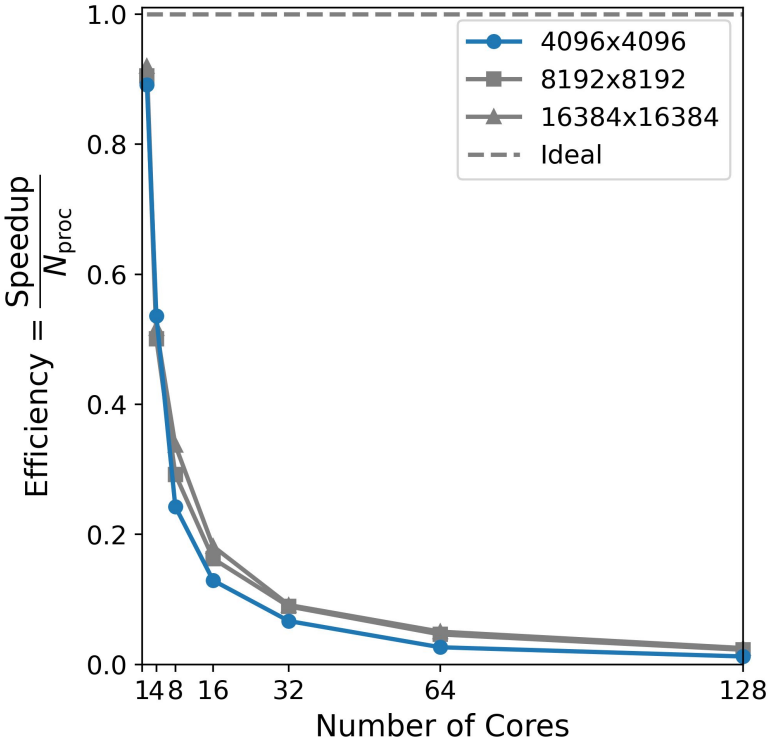




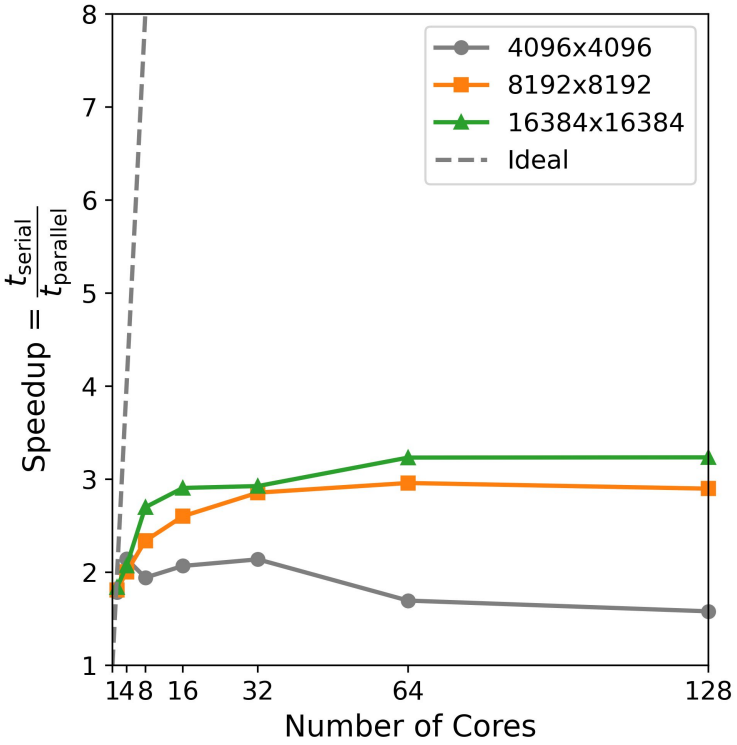
Shared Memory, Single Node CPU



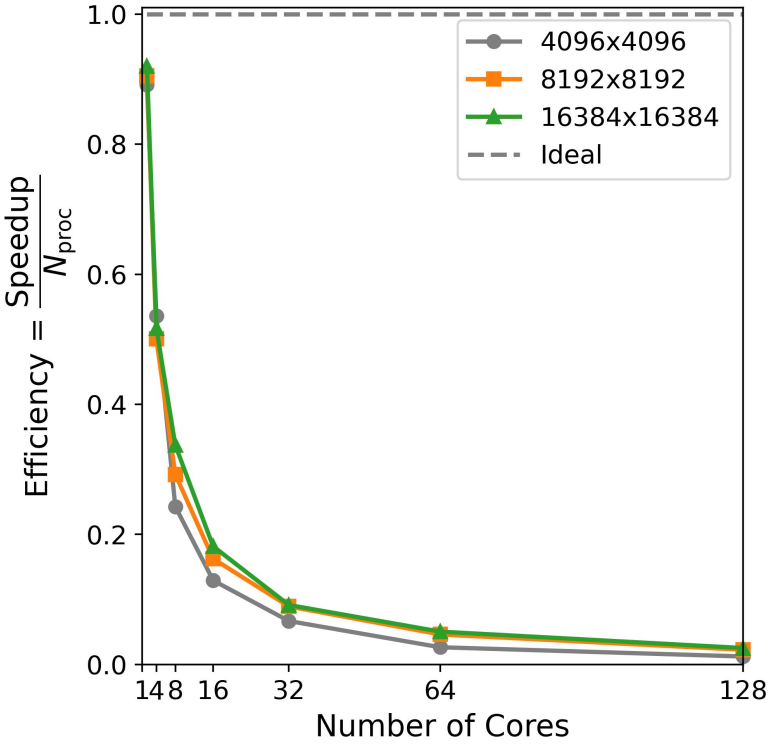
x2



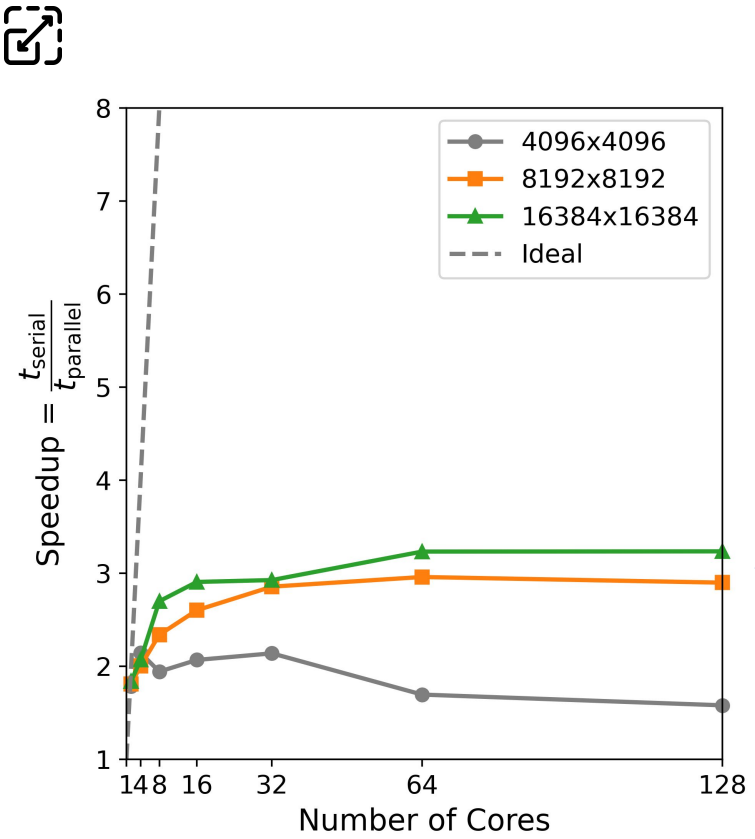
Shared Memory, Single Node CPU

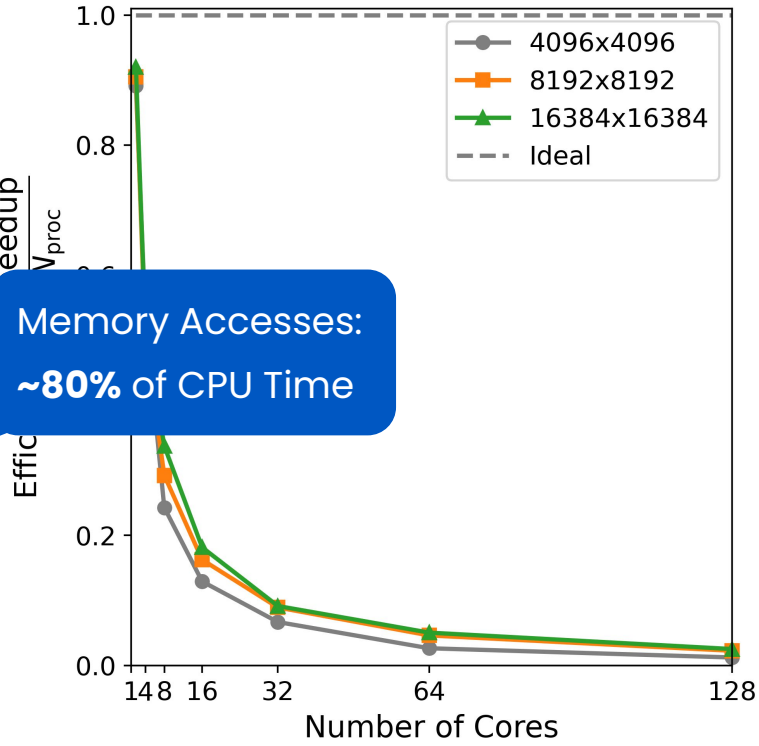


x3



Shared Memory, Single Node CPU





Memory Accesses:
~80% of CPU Time

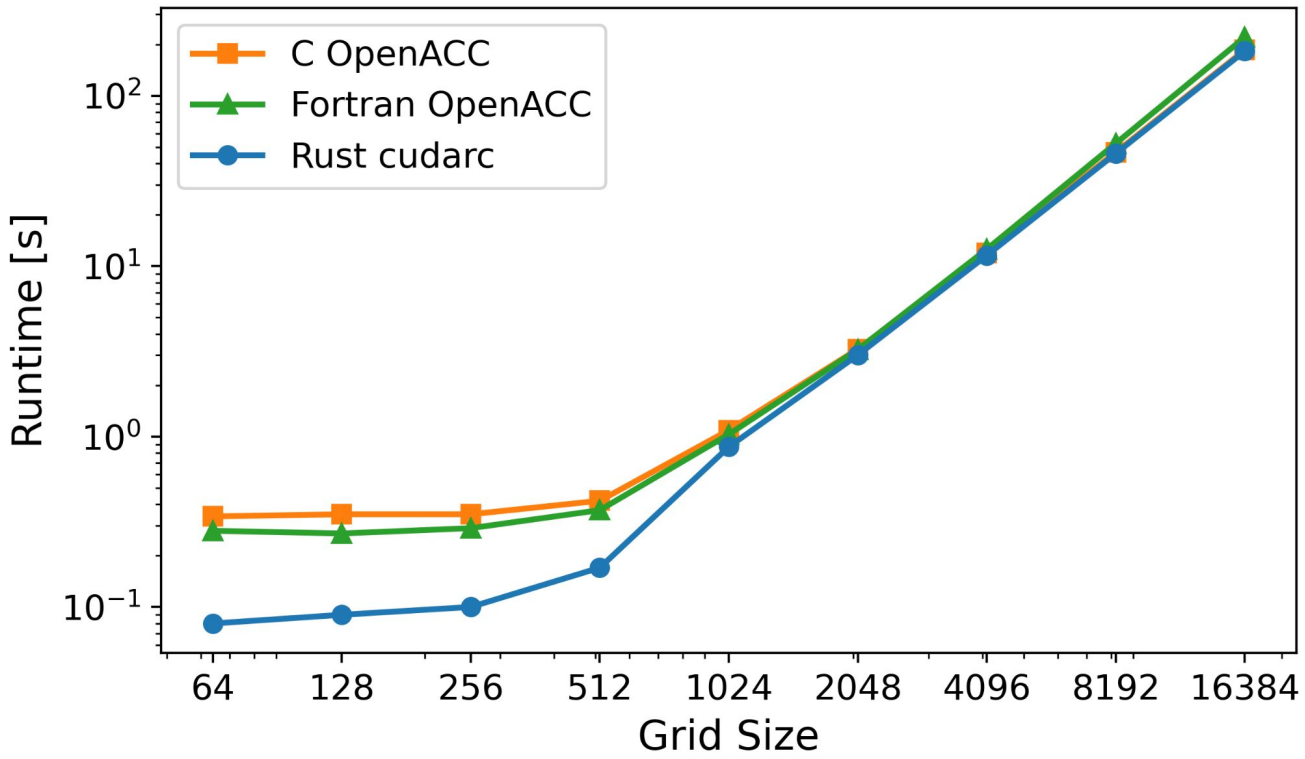
Rust GPU Support Options

Crate	Kernel language	Portability	Stage of Development
cudarc	CUDA C	NVIDIA only	stable
rust-cuda	Rust	NVIDIA only	early
opencl3	OpenCL C	GPU portable	stable
CubeCL	Rust	GPU portable	early

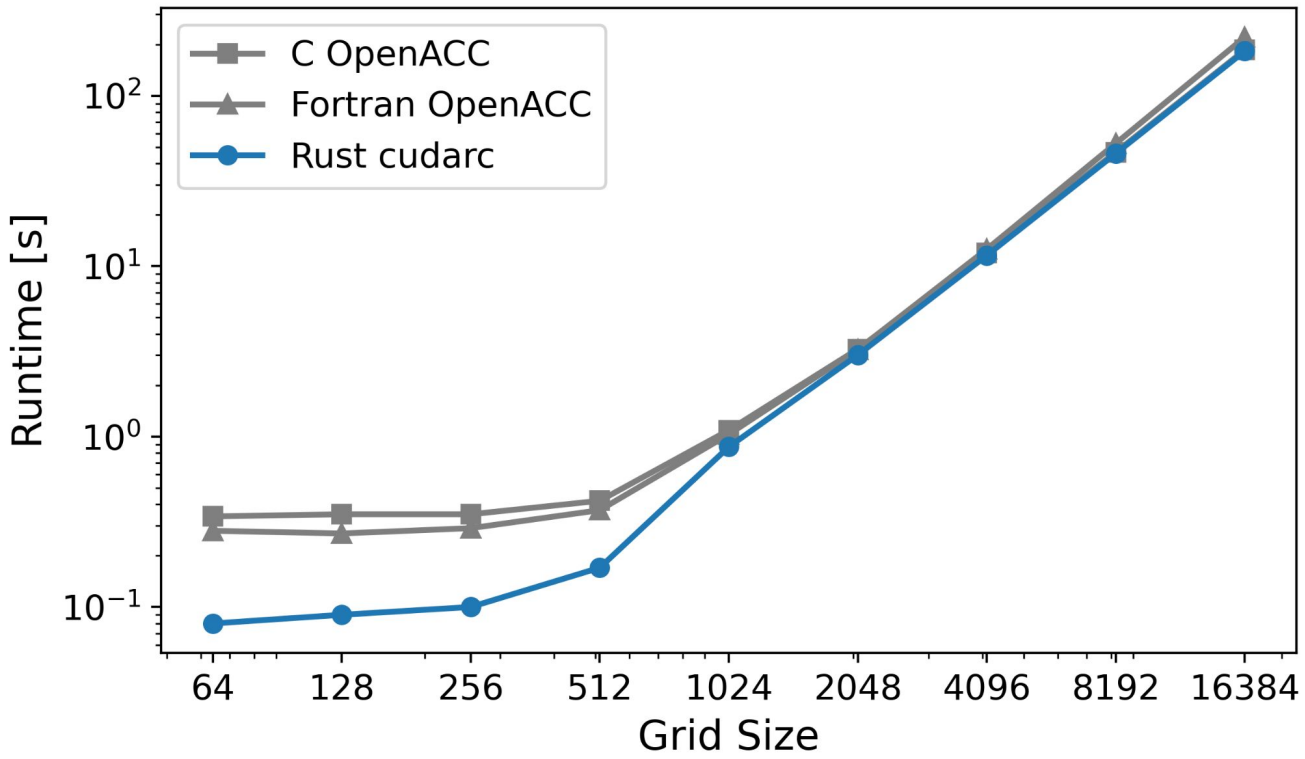
Rust GPU Support Options

Crate	Kernel language	Portability	Stage of Development
cudarc	CUDA C	NVIDIA only	stable
rust-cuda	Rust	NVIDIA only	early
opencl3	OpenCL C	GPU portable	stable
CubeCL	Rust	GPU portable	early

Single GPU


C OpenACC
Fortran OpenACC
Rust cudarc
Runtime [s]
10^2
10^1
10^0
10^{-1}
64
128
256
512
1024
2048
4096
8192
16384
Grid Size

Single GPU


C OpenACC
Fortran OpenACC
Rust cudarc
Runtime [s]
10^2
10^1
10^0
10^{-1}
64
128
256
512
1024
2048
4096
8192
16384
Grid Size

Usability

Strengths

Weaknesses

Usability

Strengths

Easy-to-use packaging

Crates

Cargo

Weaknesses

Usability

Strengths

Easy-to-use packaging

Compile-time errors

Weaknesses

```
error: could not compile `project` (bin
"project") due to 1 previous error
```

Usability

Strengths

- Easy-to-use packaging
- Compile-time errors
- Quality error messages

Weaknesses

```
12 |     let x = 5;
   |         - first assignment to `x`
14 |     x = 10;
   |     ^^^^^^ cannot assign twice to
immutable variable
   |
help: consider making this binding mutable
   |
12 |     let mut x = 5;
   |         +++
```

Usability

Strengths

- Easy-to-use packaging
- Compile-time errors
- Quality error messages

Weaknesses

- Ownership system

```
{
    let a = 5;  // a is owner
    let b = a;  // b is owner
}
                // out of scope,
                // value dropped
```

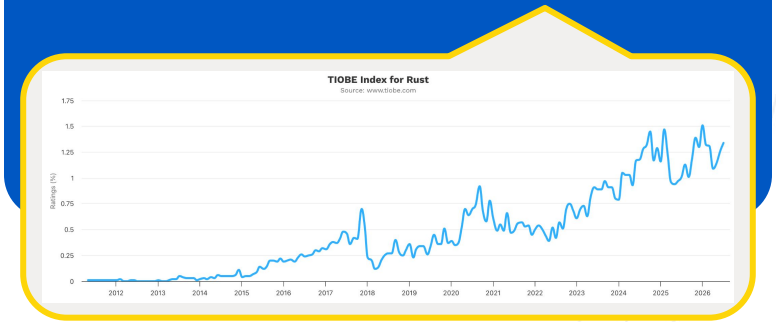
Usability

Strengths

- Easy-to-use packaging
- Compile-time errors
- Quality error messages

Weaknesses

- Ownership system
- Young language



Usability

Strengths

- Easy-to-use packaging
- Compile-time errors
- Quality error messages

Weaknesses

- Ownership system
- Young language

Conclusion

Main Takeaways

- Developed Rust implementation of shallow water model
- CPU and GPU performance comparable to C and Fortran, faster than Python
- Rust is a promising candidate for high-performance, scientific computing

Conclusion

Main Takeaways

- Developed Rust implementation of shallow water model
- CPU and GPU performance comparable to C and Fortran, faster than Python
- Rust is a promising candidate for high-performance, scientific computing

Future Work

- Research and refine shared memory code to improve speedup
- Explore and develop distributed memory code in Rust using MPI
- Further explore GPU support in pure Rust

Conclusion

Main Takeaways

- Developed Rust implementation of shallow water model
- CPU and GPU performance comparable to C and Fortran, faster than Python
- Rust is a promising candidate for high-performance, scientific computing

Future Work

- Research and refine shared memory code to improve speedup
- Explore and develop distributed memory code in Rust using MPI
- Further explore GPU support in pure Rust

Link to repository: https://github.com/NCAR/SWM/tree/swm_rust

Backup Slides

Running the Code

Rust

```
$ M=2048 N=2048 cargo run --release
   Compiling project v0.1.0 ($USER/project-dir)
    Finished `release` profile [optimized] target(s) in 0.47s
     Running `target/release/project`
```

C

```
$ gcc -DM=2048 -DN=2048 -O3 project.c -o project
$ ./project
```

Fortran

```
$ gfortran -DM=2048 -DN=2048 -O3 project.f90 -o project
$ ./project
```

Python

```
$ python project.py --M=2048 --N=2048
```

Usability

Strengths

Easy-to-use packaging

- Crates have good documentation
- Cargo, Rust's build system and package manager, makes it easy to add crates and manage dependencies
- Similarity to other popular packages, e.g. ndarray ↔ NumPy

Compile-time errors

- Debugging happens before runtime
- Avoids seg faults

Quality error messages

- Understandable messaging, useful tips to debug

Weaknesses

Ownership system

- Memory is managed through a set of rules called ownership
- More involved than some other memory-safe languages
- Learning this new system may increase time to code development

Young language

- Still developing, software community growing
- Fewer examples, crates, documentation than some other popular languages/packages

Ownership Rules

1. Each value in Rust has a owner.

2. There can only be one owner at a time.

3. When the owner goes out of scope, the value will be dropped.