A Performance and Usability Study of the Rust Programming Language

Riley Fisher[1,2]
Mentors: Lalo Torres[1], Ren Stengel[1], John Dennis[1]
[1]National Center for Atmospheric Research, [2]University of Colorado Boulder

Introduction

Motivation

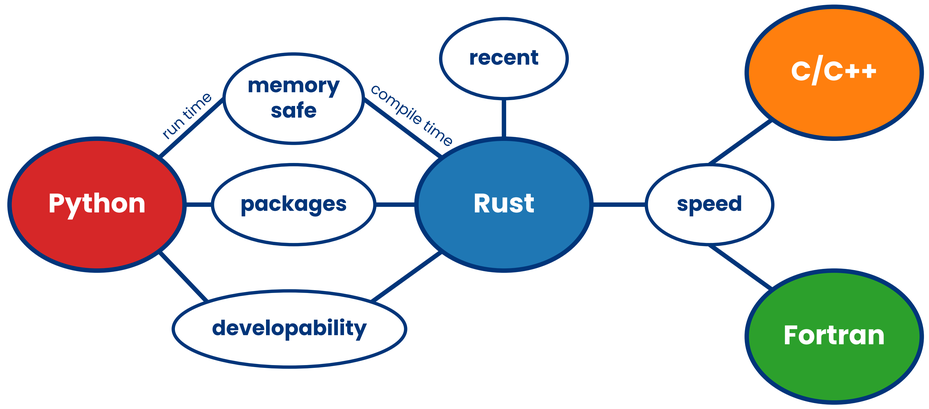
Current state
NSF NCAR develops and maintains community models that are critical to advancing our understanding of the Earth system. Collectively, these models contain millions of lines of code of which most are currently unable to take advantage of modern hardware architectures or performance libraries.

Better practices
As a leader in the field, NCAR explores and evaluates which changes in programming practices are essential for keeping up with modern computing standards. Of the many aspects to consider, the choice of programming language is the foundation for creating models. We want to prioritize developability and usability when choosing a programming language.

Maintain performance
The chosen programming language should match or exceed the performance of current implementations.

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

What is Rust?



GPU Support

GPU support for Rust is an actively developing area. Most developed crates use Rust code on the CPU to call GPU kernels from other toolkits. Writing native Rust to run on the GPU directly is still in its early stages.

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

Usability of Rust

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

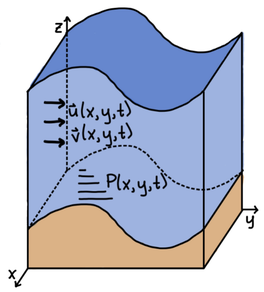
Ownership system
- Memory is managed through a set of rules called ownership
- More involved than some other memory-safe languages
- Learning this new system may increase time to code development

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

Explanation of Problem

Choice of Model: Shallow Water Equations

The shallow water equations model **velocity** and **pressure** of an incompressible fluid. The fluid is "shallow", i.e. the vertical scale is much smaller than the horizontal scale. These equations are commonly used to model ocean phenomena, such as tsunamis and storm surges.



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

Study Setup

Programming Languages
- Rust
- C
- Python
- Fortran

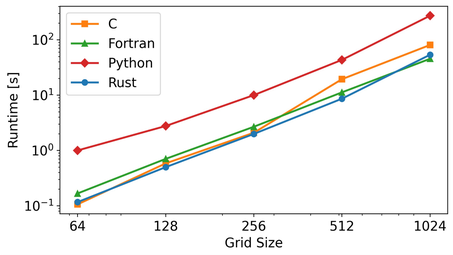
Implementations
CPU / GPU
Serial / Parallel

Performance Criteria
- Runtime
- Strong Scaling
- Usability

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

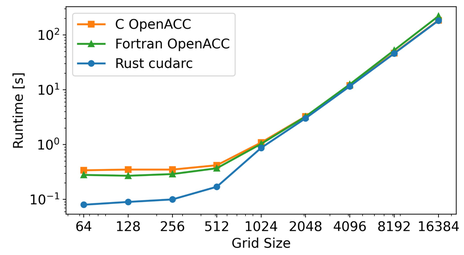
Performance Results

Serial, single core CPU



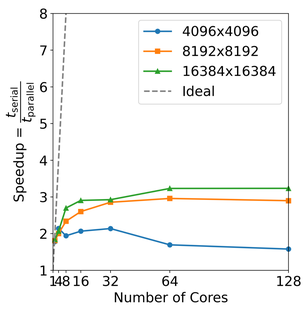
Speed: Rust ndarray ~ C ~ Fortran < Python NumPy

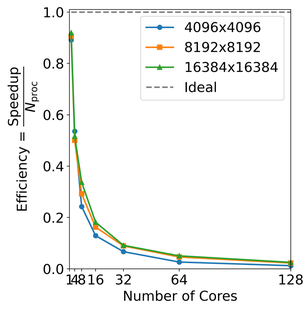
Single GPU



Speed: Rust ~ C OpenACC ~ Fortran OpenACC

Shared Memory, single node CPU





Speedup: 2-3x speedup as cores increase

Efficiency: diminishes quickly as cores increase

Memory Accesses: ~80% of CPU time, code is memory-bound

Conclusion

Main Takeaways

- Developed Rust implementation of shallow water model
- CPU/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 in Rust using MPI
- Further explore GPU support in Rust

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.