When unsafe code is slow
Automatic Differentiation in Rust

Manuel Drehwald (University of Toronto)

Motivation

- How good is idiomatic Rust?
- Is safety a performance “bug”, or a feature?
- Focusing on HPC, Scientific Computing, and ML

Unsafe Superpowers

“You can take five actions in unsafe Rust that you can’t in safe Rust”[1]

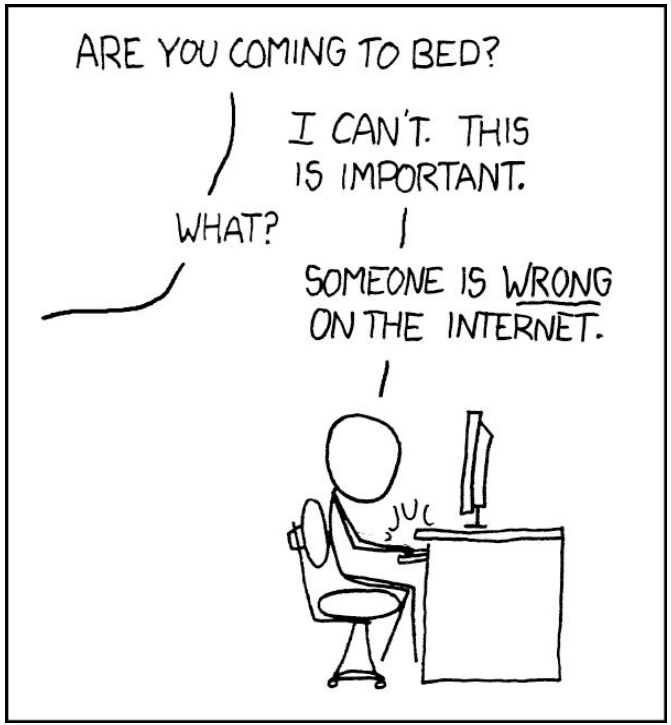
- **Dereference a raw pointer**
- **Call an unsafe function or method**
- Access or modify a mutable static variable
- Implement an unsafe trait
- Access fields of a union

We don’t track (un)safety further

[1] https://doc.rust-lang.org/book/ch19-01-unsafe-rust.html#unsafe-superpowers

False compliments

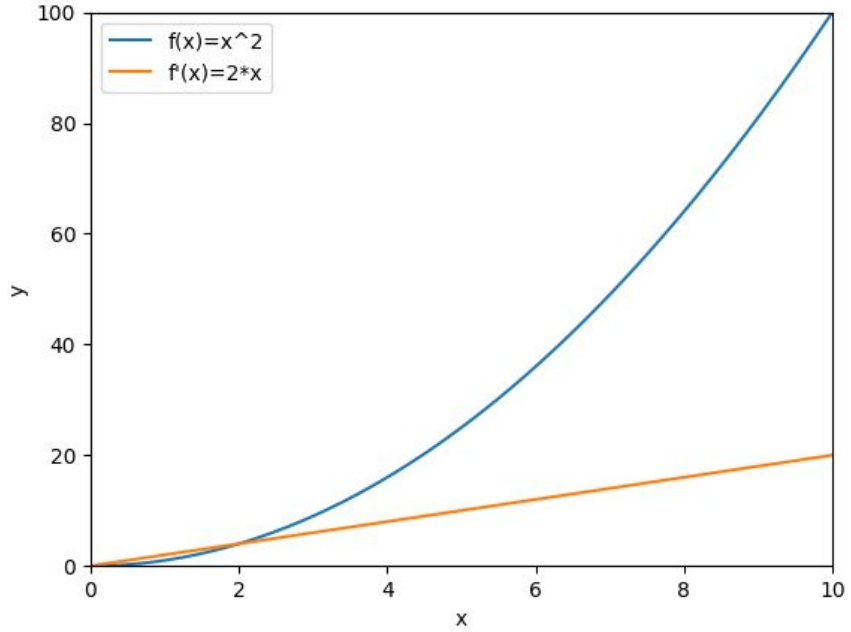
- Safety costs are too high
- But you can write it unsafe like C to be fast :)



https://xkcd.com/386/

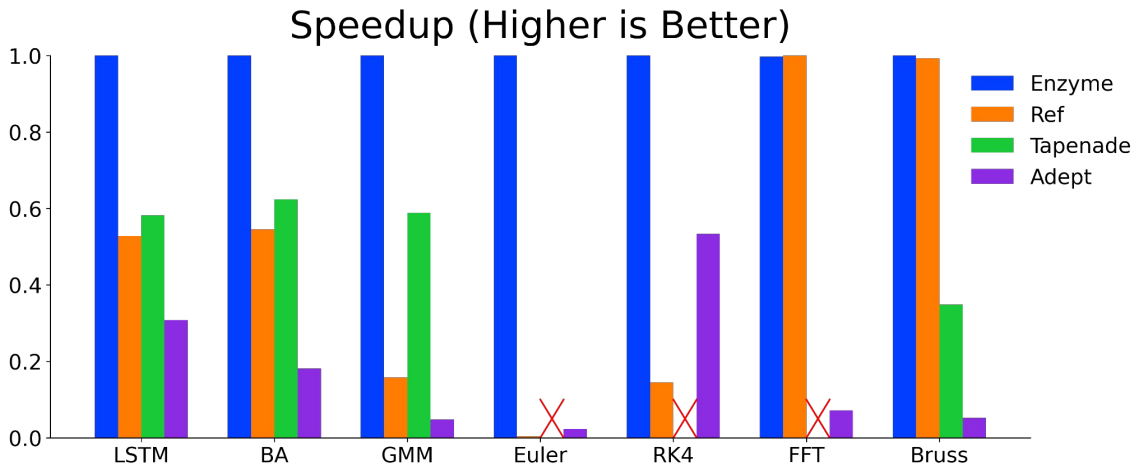
What is this Automatic Differentiation (AD)?

- Derivatives, but for Code



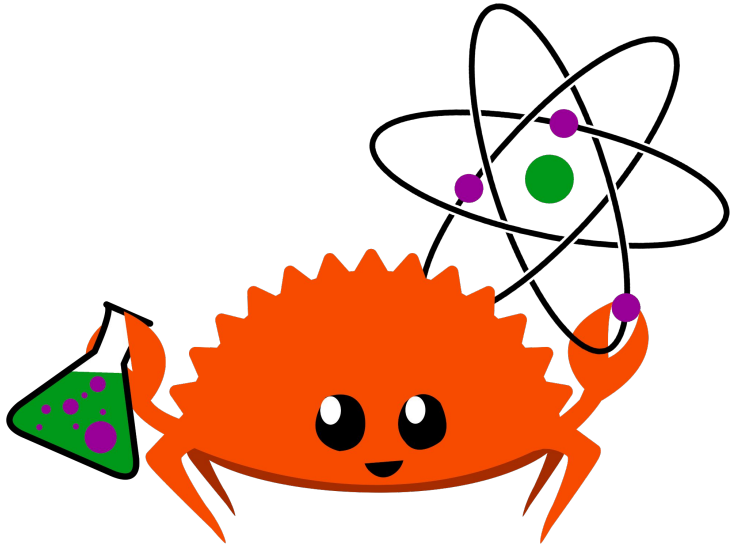
Automatic Differentiation for LLVM

- Performing AD after optimization yields a 4.2× speedup
- Rust IR is too unoptimized for AD



Selling it to Rust

- Climate Simulations
- ODE Solvers
- Differentiable Rendering
- Neural Networks
- Mechanical Engineering
- Quantum Computing
- Molecular Forces in Chemistry
- …



https://scientificcomputing.rs/

The current status

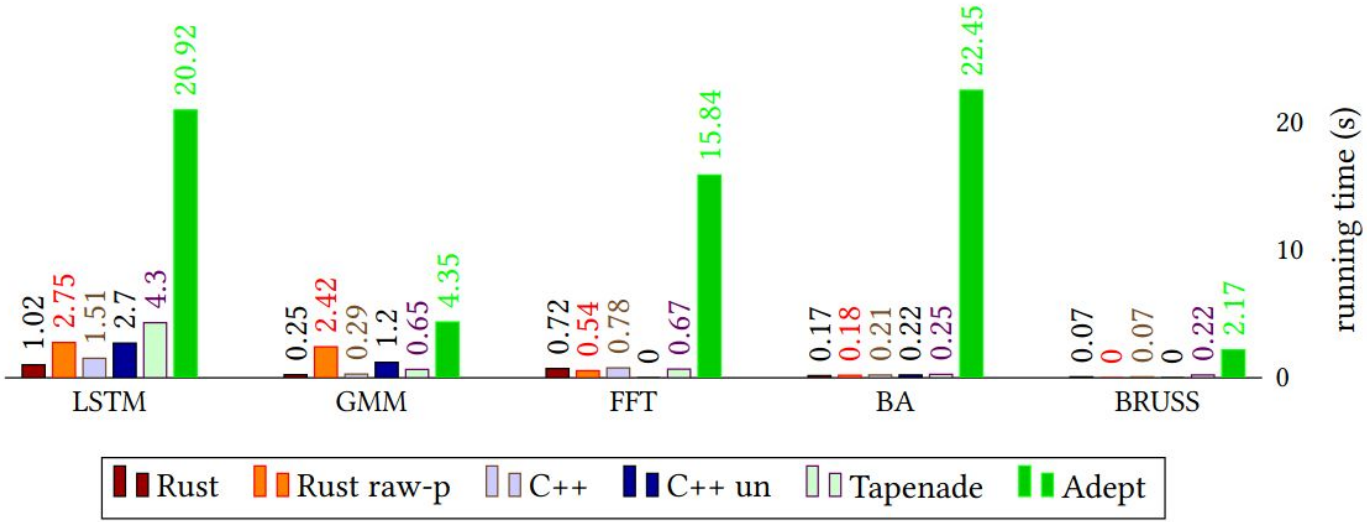
- Supported by a Rust Foundation Fellowship
- Mostly merged!

So unsafe code is slow?

- We have 5 Benchmarks
- C++ vs. Rust
- With and without noalias
- Two competing tools for C++ (Tapenade, Adept)

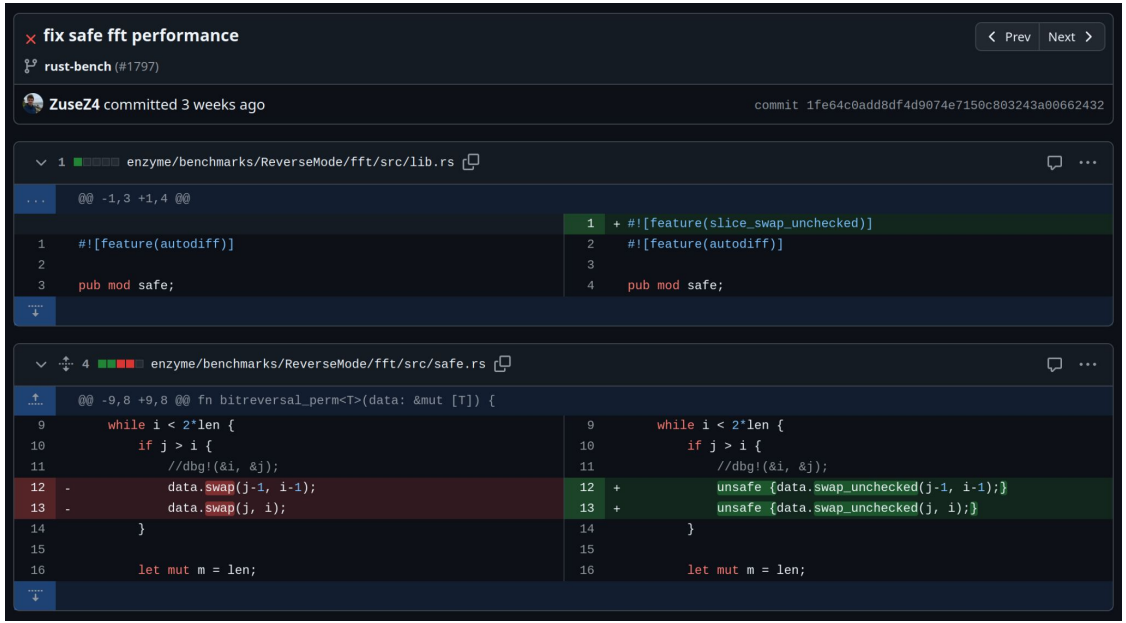
How bad can it be?

- All C++/Rust versions on pair without AD
- Disclaimer: Early numbers



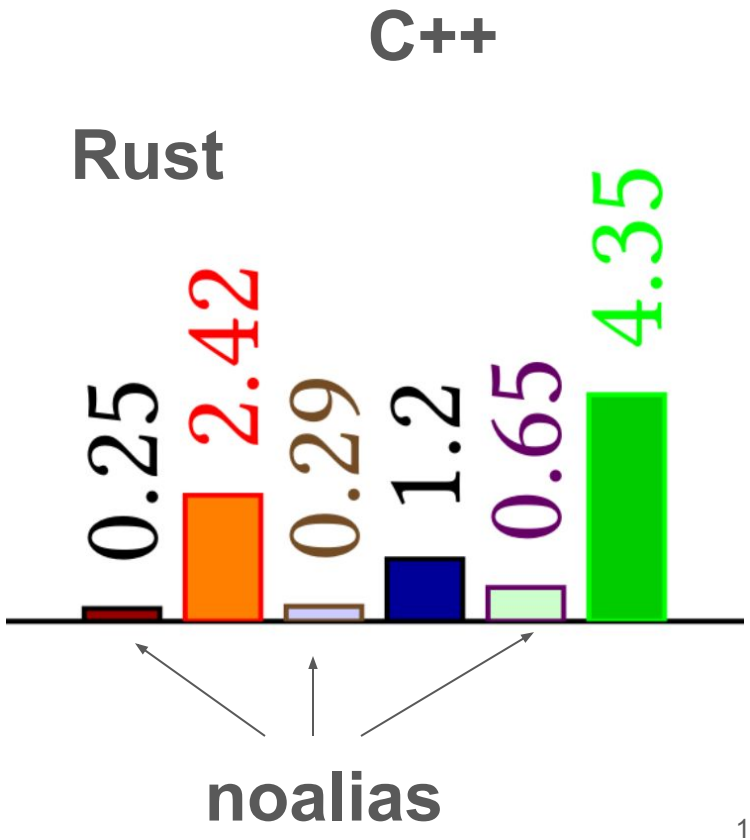
A closer look at FFT

- Bounds-checking for runtime sizes is hard
- Unsafe brings us ahead
- ~40% perf. improvement



A closer look at GMM

- First 3 spots use noalias Information
- 10x performance penalty for Rust
- 4x performance penalty for C++



Why noalias?

- (Reverse-Mode) Autodiff doubles the function length

1. “Mirror” the original function
2. Cache variables in the original (forward) pass
3. Reload or Recompute variables in the new (reverse) pass

```
subroutine  dgemm (transa, transb, m, n, k, alpha, a, lda, b, ldb, beta, c, ldc)
            DGEMM
```

https://netlib.org/lapack/explore-html/de/d6a/group__blas__top.html

Compile Times

Challenges for Automatic Differentiation

- We need Type Info for correct AD
- C++ has TBAA.

```
void f(void* dst, void* src) { memcpy(dst, src, 8); }
```

```
// Assume double inputs
∇f(double* dst, double* ddst,
   double* src, double* dsrc) {
  // Forward pass
  memcpy(dst, src, 8);
  // Reverse pass
  dsrc[0] += ddst[0];
  ddst[0] = 0;


}
```

```
// Assume float inputs
∇f(float* dst, float* ddst,
   float* src, float* dsrc) {
  // Forward pass
  memcpy(dst, src, 8);
  // Reverse pass
  dsrc[0] += ddst[0];
  ddst[0] = 0;
  dsrc[1] += ddst[1];
  ddst[1] = 0;
}
```

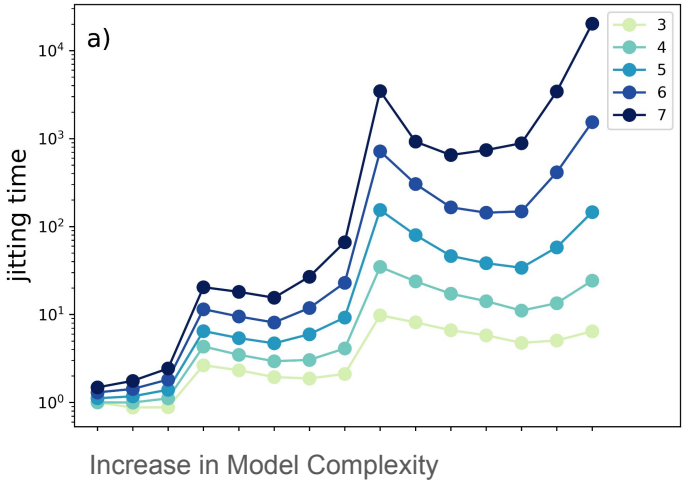
Figure 2. ***Top:*** Call to `memcpy` for an unknown 8-byte object. ***Left:*** Gradient for a `memcpy` of 8 bytes of double data. ***Right:*** Gradient for a `memcpy` of 8 bytes of float data.

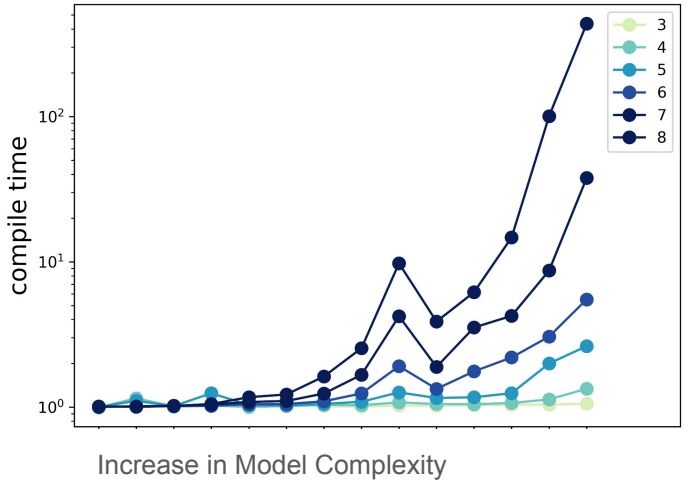
Compile time fixes for Rust

- MIR = Rust Mid-Level IR

Compile times for JAX vs Rust

- Normalized time against the smallest Chemistry Model
- JAX Model of degree 8 benchmarks ran Out-Of-Memory (>50k LoC)





Can C++ do better?

- 5 functions from our Chemistry Model
- 500 lines each
- C++: std::vector
- Rust: std::vec::Vec (vec[] and vec.unchecked_get())

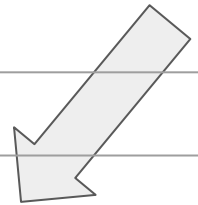
```cpp
void f_1(std::vector<float>& vec) {
    vec[500] = vec[457] + vec[481] - vec[212];
    vec[501] = vec[471] - vec[31] - vec[335];
    vec[502] = vec[137] * vec[418] - vec[113];
    vec[503] = vec[97] * vec[436] - vec[403];
    vec[504] = vec[277] - vec[339] + vec[392];
    vec[505] = vec[293] + vec[344] * vec[2];
    vec[506] = vec[22] * vec[330] * vec[4];
    vec[507] = vec[364] + vec[156] - vec[197];
```

```rust
fn f_1(vec: &mut [f32]) {
    vec[500] = vec[228] + vec[308] + vec[39];
    vec[501] = vec[59] - vec[74] - vec[368];
    vec[502] = vec[41] * vec[133] + vec[40];
    vec[503] = vec[216] - vec[167] * vec[246];
    vec[504] = vec[147] - vec[342] * vec[430];
    vec[505] = vec[503] - vec[29] * vec[31];
    vec[506] = vec[129] - vec[73] + vec[401];
    vec[507] = vec[168] * vec[35] * vec[167];
```

Can C++ do better?

- 5 functions
- 500 lines each

50% LLVM post-opt (Vectorizer?)



Compile Time	C++	Rust	Rust unchecked_get()
Release build	131s	87s	29.3s
Debug build	> 1hr	8.3s	130s
Baseline (release)	3.3s	6.0s	9.2s

Correctness

Bounds checking for shadow memory?

- We assert that shadow slices are long enough.
- We don’t handle vectors or indirections yet. **UB**

```
fn main() {
    let x = vec![1.0; 10];
    let mut too_short = vec![0.0; 9];
    df(&x, &mut too_short);
}
```

Mutability of shadow memory

- Enzyme can overwrite immutable Rust types. **UB**
- We don’t find it in structs

```
struct evil {
    value: f32,
    const_data: &'static f32,
}
```

- But in Slices

```
warning: reading from a `Duplicated` const ref is unsafe
 --> src/main.rs:6:1
  |
6 | fn f(x: &[&f32], out: &mut f32) {
  | ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

Passing in wrong enum Variants?

Enzyme can overwrite wrong Rust types. **Sometimes UB**

```
enum Evil {
    Value(f32),
    Fineish(i32),
    Bad(*mut f32),
    Worse(Option<&f32>),
}
```

Questions?