College of Engineering and Computing
ELECTRICAL AND
COMPUTER ENGINEERING
George Mason University

ECE 555 Group Presentation

Igor Semyonov Jordan Carnes Robert Laverne Griffin
George Macon University, Department of Electrical and Computer Engineering

April 23, 2025

Why not Just Use C?

It should not be used for production.
See `https://veresov.pro/cmustdie/`

Why Rust
Type System

Listing 1: Cats

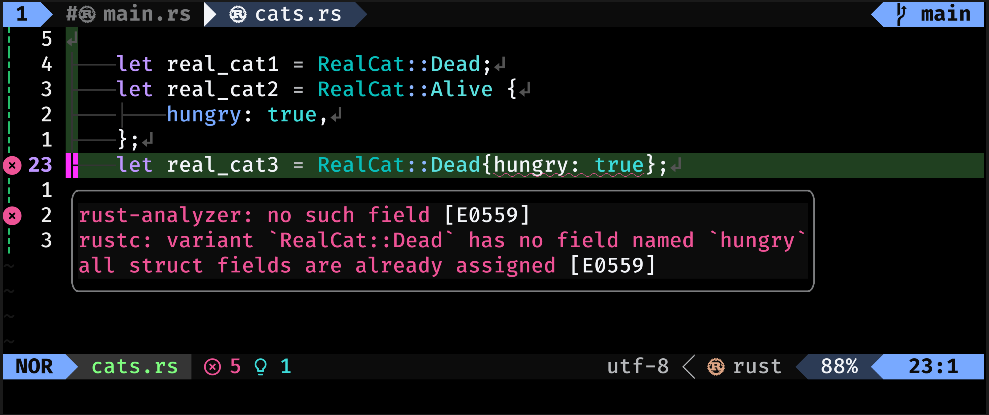
```
struct FakeCat {
    alive: bool,
    hungry: bool,
}

enum RealCat {
    Alive {
        hungry: bool,
    },
```

Impossible states can be encoded in the type system, which leads to compile time errors, instead of runtime checks.

Listing 2: Unphysical (zombie) Cat

```
    let fake_cat = FakeCat {
        alive: false,
        hungry: true,
    };
```

Why Rust
Type System



```

    let real_cat1 = RealCat::Dead;
    let real_cat2 = RealCat::Alive {
        hungry: true,
    };
    let real_cat3 = RealCat::Dead{hungry: true};
```

```
rust-analyzer: no such field [E0559]
rustc: variant `RealCat::Dead` has no field named `hungry`
all struct fields are already assigned [E0559]
```

Figure 1: Error when declaring an unrealistic cat

Why Rust
Safety

- Result: Errors are values, i.e., no nidden control flow
- Functions that can fail return Result and this must be explicitly handled.

- No manual memory management
- Values are dropped once they go out of scope

Why Rust
Ergonomics

High level abstractions that enable multithreading without the error-prone approach of pthreads in C.

Why Rust
Single threaded bsort

Listing 3: Single threaded bsort

```
a.chunks_mut(split_length)
    .for_each(bsort);
```

- a is a vector of numbers
- `chunks_mut` slices the vector into non-overlapping slices of the given length
- `for_each` iterates over the chunks, running the provided function with each chunk as the input.

Why Rust
Concurrency is easy while avoiding race conditions

Listing 4: Example of easy parallelization

```
    a.chunks_mut(split_length)
        .par_bridge()
        .for_each(bsort);
```

- Chunks is known to split a vector into distinct slices
- Hence we can safely send each chunk to a different thread

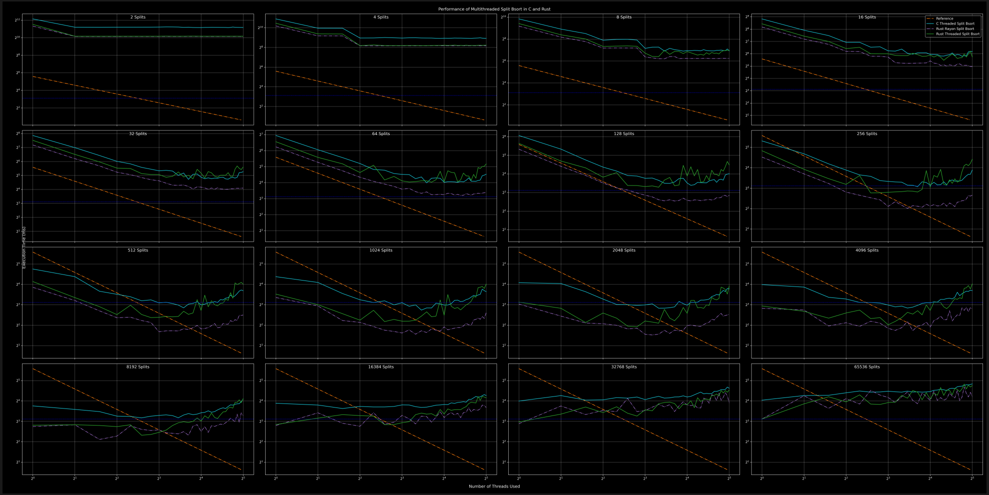


Figure 2: Preformacne results from bsort project. Note that the Rust reference sort is not plotted as it was under 1ms, compared to 8ms for the C qsort.

Why Rust
Zero Cost Abstraction

All these features are provided at compile time and are optimized away when compiling so that they do not affect performance at run time.

Rust Ownership and Borrowing
Problems with shared memory access in Rust

- Ownership Rules[1]
  - Each value has an owner
  - There can only be one owner at a time
  - When the owner goes out of scope, the value will be dropped.

- Borrow Rules[2]
  - At any given time, you can have either one mutable reference or any number of immutable references.
  - References must always be valid.

C, and especially CUDA, style shared mutable memory access violates the ownership and borrow rules.
At this time, we therefore need to use unsafe Rust when writing and calling GPU kernels.

[2]Klabnik et al. 2024, Ch04-01.
[2]Klabnik et al. 2024, Ch04-02.

Rust and CUDA
Current options

- Rust GPU for vendor agnostic GPU programming
- Rust CUDA for targeting NVIDIA GPUs and their specific libraries
- Vulkano
- wgpu
- Miniquad
- Sierra
- Glium
- Ash
- Erupt

Rust and CUDA
Description of how the Rust-CUDA crate works

- **Purpose:** Enable GPU programming in Rust by writing CUDA kernels directly in Rust.
- Need the CUDA SDK and LLVM7.x.

Rust and CUDA
Focusing on Rust CUDA

Currently being rebooted and is in active development.
Uses nvidia's nvvm tool which is built on LLVM 7.

What is a compiler?
Translation pipeline

Front End

- Lexical analysis: source $\rightarrow$ tokens (identifiers, literals, operators)
- Syntax & semantic analysis: tokens $\rightarrow$ AST, type checking
- Emit language independent IR

Optimization

- IR level passes: constant folding, dead code elimination, algebraic simplification
- Advanced transforms: function inlining, loop unrolling, vectorization
- Profile or link time optimizations for extra performance

Back End

- Lower optimized IR to machine instructions (instruction selection & scheduling)
- Register allocation, calling-convention handling
- Emit assembly or object code ready for linking

The problem LLVM solves

Description of LLVM and it's intermediate representation and how this has enabled much easier language development.

NVVM

How NVVM works

Rust CUDA and NVVM

How NVVM is used in Rust CUDA

Demo Implementation

Fractals
Mandelbrot and Burning Ship

$c \in \mathbb{C}$ is in the mandelbrot set if the sequence $\{z_n\}$ converges.

$$z_{n+1} := z_n^2 + c \qquad z_0 = 0$$

The Burning Ship fractal is defined similarly but the sequence is

$$z_{n+1} := (|\mathrm{Re}(z_n)| + |\mathrm{Im}(z_n)|i)^2 + c \qquad z_0 = 0$$

Mandelbrot Kernel

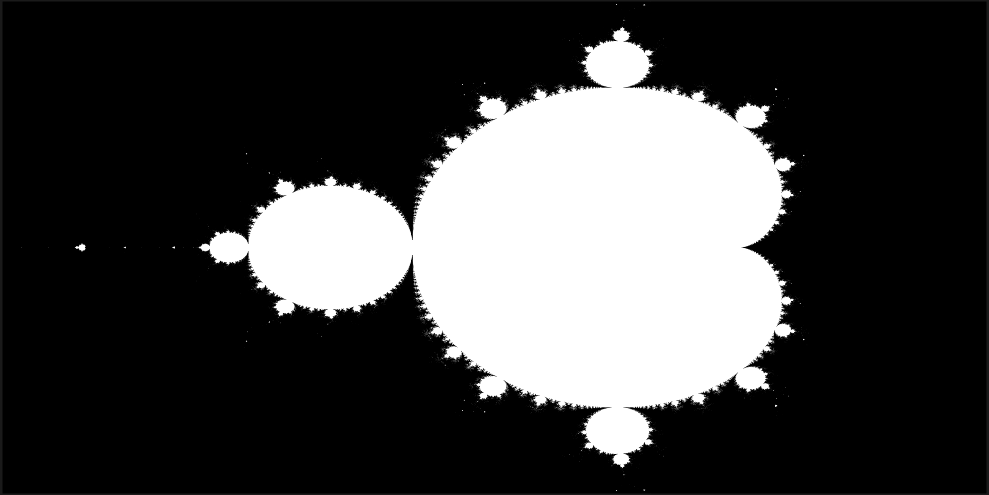
```
pub unsafe fn mandelbrot(
    n_re: usize,
    n_im: usize,
    re_min: f32,
    re_max: f32,
    re_range: f32,
    im_min: f32,
    im_max: f32,
    im_range: f32,
    zn_limit: u32,
    out: *mut u8,
) {
    let idx_linear = thread::index() as usize;
    let idx = thread::index_2d();
    let idx_re = idx[1];
    let idx_im = idx[0];

    let c_re = re_range
        * (idx_re as f32 / (n_re - 1) as f32)
        + re_min;
    let c_im = im_range
        * (idx_im as f32 / (n_im - 1) as f32)
        + im_min;
    let mut z_re = c_re;
    let mut z_im = c_im;
    for _ in 0..zn_limit / ZN_SKIP {
        for _ in 0..ZN_SKIP {
            (
                z_re, z_im,
            ) = (
                z_re * z_re - z_im * z_im +
                     c_re,
                2.0 * z_re * z_im + c_im,
            );
        }
        if z_re * z_re + z_im * z_im > 4.0 {
            let elem = &mut *out.add(
                 idx_linear);
            *elem = 255;
            break;
        }
    }
}
```

Launching the Kernel

```
    unsafe {
        launch!(
            module.mandelbrot<<<grid_size, block_size,
                  0, stream>>>(
                N_RE,
                N_IM,
                re_min,
                re_max,
                re_range,
                im_min,
                im_max,
                im_range,
                zn_limit,
                out_gpu.as_device_ptr(),
            )
        )?;
    }
```

- Fiarly similar to launching a kernel in C.
- Must be used inside an `unsafe` block since the kernel itself is an unsafe function.

Output Image



Fractals
Timing Results for 1 frame on CPU and GPU

I will add the single, frame timing results here

Live Demo

I plan to be sharing my screen for the presentation and will switch to a live demo here.

This is a slide
With a subtitle

This is some text in a column. Could be a figure instead.

- This is a list
- It is an itemized one
- Hence the bullets

References

Klabnik, Steve et al. (2024). *The Rust Programming Language*. No Starch Press. ISBN: 978-1-63190-308-0. URL: https://doc.rust-lang.org/book/.