
# ECE 555 Group Presentation

Igor Semyonov  Jordan Carnes  Robert Laverne Griffin

George Macon University, Department of Electrical and Computer Engineering

April 4, 2025

# Why not Just Use C?

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

# Why Rust?
## Type System

insert real/fake cat example from
`https://www.youtube.com/watch?v=z-0-bbc80JM&t=514s`

Insert image of my IDE showing an error such as attempting to use a value after it is moved out of scope.

# Why Rust?
## Ergonomics while remaining fast

Here I may include my rust implementation of project 1 and compare it to my C version in both ergonomics, safety, and speed.

# Why Rust?
## Single threaded bsort

**Listing 1:** Single threaded bsort

```
1  a.chunks_mut(split_length)
2      .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 2:** Example of easy parallelization

```
1  a.chunks_mut(split_length)
2      .par_bridge()
3      .for_each(bsort);
```

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

# Rust Ownership and Borrowing
## Problems with shared memory access in Rust

- Ownership Rules[1, Ch04-01]
  - 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[1, Ch04-02]
  - 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.

# Rust and CUDA
## Current options

- Rust GPU
- Rust CUDA

Maybe mention other options for GPU programming in rust, like any rust support for ROCm, HIP, intel gaudi, or other.

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

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

# What is a compiler?

Description of problems when going straing from source code to machine code.

# 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

$c \in \mathbb{C}$ 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

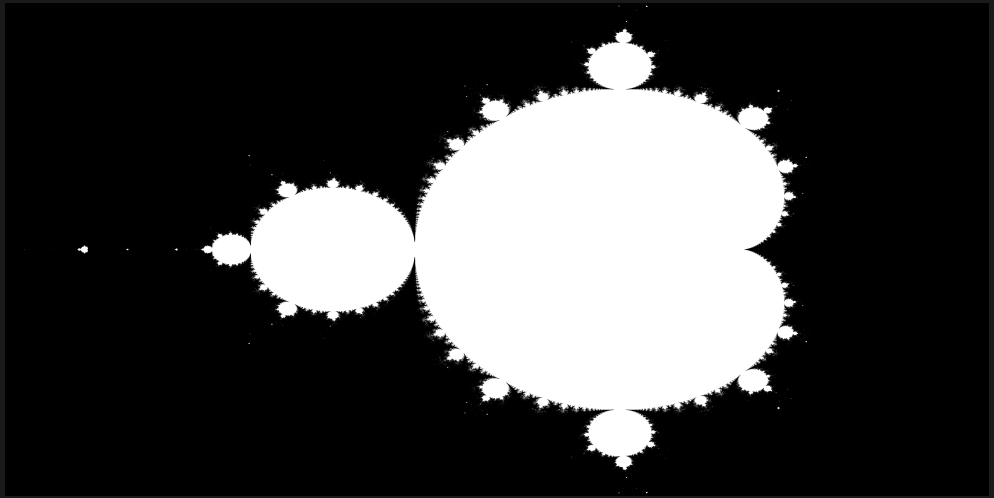
```rust
pub unsafe fn mandelbrot(
    n_re: usize,
    n_im: usize,
    re_min: f32,
    re_max: f32,
    re_range: f32,
    im_min: f32,
    im_max: f32,
    im_range: f32,
    zn_limit: u32,
    out: *mut u8,
) {
    let idx_linear = thread::index() as usize;
    let idx = thread::index_2d();
    let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

```
1     unsafe {
2         launch!(
3             module.mandelbrot<<<grid_size, block_size,
                  0, stream>>>(
4                 N_RE,
5                 N_IM,
6                 re_min,
7                 re_max,
8                 re_range,
9                 im_min,
10                im_max,
11                im_range,
12                zn_limit,
13                out_gpu.as_device_ptr(),
14            )
15        )?;
16    }
```

- 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

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


S. Klabnik, C. Nichols, K. Chris, and Rust Community, *The Rust Programming Language*.
No Starch Press, 2024.