Advanced Rust 2026 - Lab 1: Parallel Model and Deterministic Execution

Lukas Hozda

Spring 2026

Lab Goals

1. Practice deterministic parallel design under realistic constraints.
2. Build confidence with scoped threads, channels, and aggregation order.
3. Prepare directly for homework pair A1.

Time Plan (90 min)

1. 10 min - setup and quick recap
2. 25 min - exercise 1 (scoped parallel map)
3. 25 min - exercise 2 (worker queue with deterministic output)
4. 20 min - exercise 3 (debugging nondeterministic behavior)
5. 10 min - review and Q&A

Setup

1. Use Rust stable 1.88.
2. Create a new crate: `cargo new ar2026-lab1`
3. Work in separate binaries for each exercise.

Exercise 1: Scoped Parallel Map

Objective

Implement a function that maps integers in parallel and returns output in the original order.

Requirements

1. Use `std::thread::scope`.
2. Split work into chunks by index ranges.
3. Keep output deterministic (same order as input).
4. Handle edge cases: empty input, workers = 0, workers > len.

Suggested signature

```
fn parallel_map_ordered(values: &[i64], workers: usize) → Vec<i64>
```

Checkpoint

Print results for at least 3 different worker counts and prove outputs are identical.

Exercise 2: Deterministic Worker Queue

Objective

Build a simple worker queue using channels and aggregate results in stable order.

Requirements

1. Input tasks have fields `id` and `payload`.
2. Workers compute:
   - `sum(payload)`
   - `checksum(payload) = sum((i + 1) * value)`
3. Final output must be sorted by task id.
4. No direct printing from workers.

Suggested types

```
struct Task {
    id: i64,
    payload: Vec<i64>,
}

struct TaskResult {
    id: i64,
    sum: i64,
    checksum: i64,
}
```

Checkpoint

Demonstrate identical stdout across repeated runs.

Exercise 3: Fix a Flaky Implementation

Objective

You are given intentionally flawed code that prints from worker threads and produces random ordering. Refactor it so output is deterministic and testable.

Refactor checklist

1. Replace worker-side printing with result messages.
2. Collect all results centrally.
3. Sort before formatting final text.
4. Add tests for empty and mixed-sign data.

Debrief Questions

1. Which design choices made determinism easiest?
2. Where did ownership/lifetimes influence architecture?

3. Which parts map directly to A1A and A1B?

Homework Link

This lab directly supports:

1. `homeworks/2026/advanced/pair-01-parallel-model/hw-a-library`
2. `homeworks/2026/advanced/pair-01-parallel-model/hw-b-executable`