

Redline: Compile-Time Performance Guarantees for Rust

Tushar Sharma

ALIA Labs

`txshar@proton.me`

<https://github.com/TxsharDev/redline>

June 2026

Abstract

Rust proves memory safety at compile time. We prove performance at compile time. Redline is a Rust proc-macro that lets developers annotate functions with latency, throughput, allocation, and syscall bounds. The compiler statically analyzes every code path against a cost model. If any path can violate the bound, the code does not compile. No existing tool provides source-level, language-integrated performance bounds that block compilation.

1 Introduction

Performance bugs are found after deployment. Profilers measure running code. Benchmarks test specific inputs. Linters flag style. None of them prevent slow code from compiling. Redline takes a different approach: write the performance requirement in the source code. The compiler enforces it.

2 Design

Redline provides five constraint types: latency, throughput, allocations, syscalls, and stack size. The analysis walks the function AST via syn, counts allocations and syscalls by pattern-matching known function signatures, estimates loop costs from literal ranges (pessimistic default for unknowns), takes worst-case branches, and sums against a conservative cost model.

2.1 Cost Model

Conservative x86-64 estimates: ALU 1ns, call 5ns, HashMap lookup 25ns, allocation 50ns, mutex 100ns, format 200ns, syscall 1000ns, println 5000ns, network 10000ns. Intentionally pessimistic: better to reject fast code than accept slow code.

3 Evaluation

10 compile-time test cases via trybuild. Pass: pure math, branching, bounded loops, combined constraints, nested loops. Fail: `String::from` with `allocs=0`, `fs::read` with `syscalls=0`, 10K loop exceeding latency, `println` with `syscalls=0`, `TcpStream` with `syscalls=0`, allocations in loops exceeding limit.

4 Related Work

WCET analysis exists in embedded systems (aiT, Bound-T) but operates on binaries, not source. Clippy lints style, not performance. Criterion benchmarks, not guarantees. Effect systems (Koka) track effects but need language changes. Redline brings WCET concepts to source-level Rust as a proc-macro.

5 Limitations

AST-level only (no cross-function analysis). Conservative estimates may over-reject. Dynamic loop bounds default to 1000 iterations. No hardware calibration. v0.2: cross-function + LLVM cost model. v0.3: runtime verification.

6 Conclusion

Performance in the type system. Annotate, compile, guaranteed. No existing tool does this at the source level for a general-purpose language.

References

- [1] Wilhelm, R., et al. *The WCET Problem*. ACM TECS, 7(3), 2008.
- [2] Bauer, A., Pretnar, M. *Algebraic Effects*. JLAMP, 84(1), 2015.
- [3] Matsakis, N., Klock, F. *The Rust Language*. ACM SIGAda, 34(3), 2014.
- [4] Lattner, C., Adve, V. *LLVM*. CGO, 2004.
- [5] Brook, J. *Criterion.rs*. GitHub, 2023.