STAMP-Rust: Language and Performance Comparison to C on Transactional Benchmarks

Bench 2022

Felix Suchert, Jerónimo Castrillon

Nov. 9, 2022

Motivation

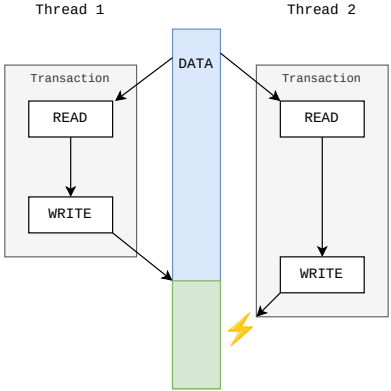
Software Transactional Memory

- Established method for state synchronization
- *But:* many frameworks written in C

Rust

Key Promises:

Safety & **Performance**



Automatic Transpilation: Language Impact

- Automatic transpilation of STAMP suite using `c2rust`

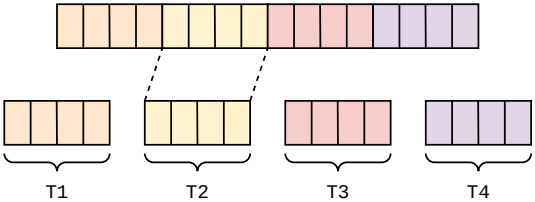
Code Quality:

- Dereferencing raw pointers
- Type casts
- Manual memory management
- Non-idiomatic
- Maintainability?

```
1  pub unsafe extern "C" fn router_solve(
2      mut argPtr: *mut libc::c_void)
3  {
4      let mut routerArgPt: *mut router_solve_arg =
5          argPtr as *mut router_solve_arg;
6      let mut routerPt: *mut router_t =
7          (*routerArgPt).routerPt;
8      let mut mazePt: *mut maze_t =
9          (*routerArgPt).mazePt;
10     let mut myPathVectorPt: *mut vector_t =
11         Pvector_alloc(
12             1 as libc::c_int as libc::c_long,
13         );
14     // ...
```

Type Level Safety

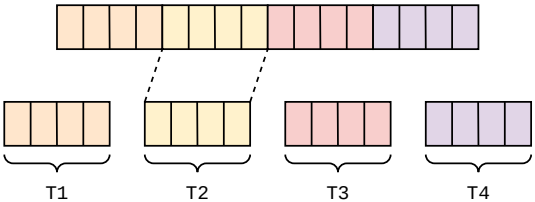
Pattern: Parallel State Accesses



- `ssca2` in C: index partitioning
- Mutable data sharing across threads is forbidden in Rust

Type Level Safety

Pattern: Parallel State Accesses



Overhead

More allocations/data movement

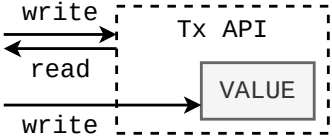
Pattern: Specialized Data Structures

- Transaction-aware HashMaps help boosting performance in C

Type Level Safety

Pattern: Violation of Transactional Model

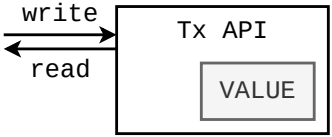
- No type guards in C implementation
- Transactional Functions are mere wrappers
- `ssca2` in C: frequent direct value accesses



Type Level Safety

Pattern: Violation of Transactional Model

- No type guards in C implementation
- Transactional Functions are mere wrappers
- `ssca2`: frequent direct value accesses
- Type system in Rust enforces usage of API

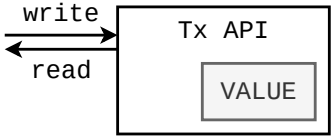


Type Level Safety

Pattern: Violation of Transactional Model

Overhead

More data duplication when working with transactional variables



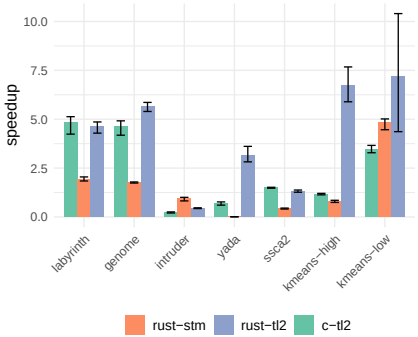
Safe Composition of Transactions

```
impl<T> for TVar<T> {
    pub fn read(&self, trans: &mut Transaction);
    pub fn write(&self, trans: &mut Transaction);
}
```

Transaction context required for variable access

Performance Results: STAMP

- Rust version implemented using `rust-stm`
- Speedup over sequential implementation
- 12 threads
- original implementation: `c-tl2`
- transpiled implementation: `rust-tl2`
- Safe Rust often significantly slower

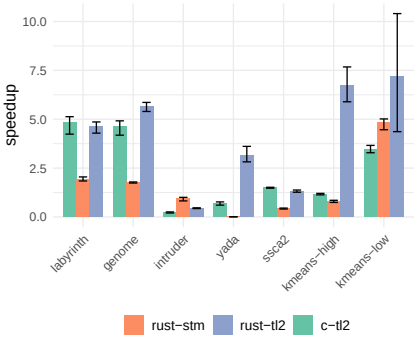


Conclusion

Do Rust's promises hold up?

- Safety: YES
- Speed: only unsafe

- `tl2` gains lot of speed from unsafe behavior



Thank you for your attention!

felix.suchert@tu-dresden.de