Data Race Detection in C/C++ Concurrent Programs

CS4560 - Parallel and Concurrent Programming

February, 2026

Project description

In this project, you will detect data races in the executions of C11 concurrent programs. [1]

Background Information

C11 concurrency

C/C++ defines the relaxed memory concurrency model which is known as the C11 concurrency model [1]. C11 has various kinds of accesses that affect shared memory concurrency. To begin with, it provides plain or non-atomic load and store accesses. In addition, C11 also has atomic accesses of four kinds: load, store, atomic update (RMW) such as compare-and-swap and atomic increment, and memory fence. Each atomic access is attached with a memory order from relaxed, acquire, release, acquire-release, sequentially-consistent.

Data race

A pair of events (a, b) is in data race if

- a and b are concurrent, that is, not related by *happens-before* relation [2, 3], and,
- a and b access the same memory location, and,
- at least one of them is a write event.

Given a data race if both memory accesses are writes then it is a write-write race. If it is between read and write accesses then we say it is a read-write race. Moreover, we may categorize data race as

1. Non-atomic-race: where at least one access in the race is non-atomic.
2. Relaxed-race: where at least one access in the race is relaxed.
3. RA-race: where at least one access in the race is non-SC access.

Execution

An execution consists of a set of events resulting from shared memory accesses or fences and relations between these events. Further details are in [1, 3, 4].

c11tester

Given a C11 program the c11tester tool [3, 4] may execute the program and generate execution traces with events and relations as discussed above. Currently, c11tester identifies data races on non-atomic accesses. In this project, you will generate the traces and identify the other types of data races.

Note: *feel free to use any other tool if you like.*

[1] The project description is subject to small changes and updates. Please contact the TAs and the teachers if you have any questions.

Projects

- **Project Relaxed-Race-Detection.** Detect relaxed-race.
- **Project RA-Race-Detection.** Detect RA-race.

Roadmap for the project:

The project involves the following steps:

- Set up the c11tester tool. It is available at `https://brightspace.tudelft.nl/d2l/le/content/680657/viewContent/4088449/View`.
- Write C11 test programs.
- Generate execution traces from C11 programs (and write it in a file).
- Develop the algorithms for data race detection on the generated trace.
- Implement the algorithm to check if an execution contains a data race. The implementation can be independent of the c11tester tool.
- Evaluate on the c11tester benchmarks*.

Note

- You may use c11tester inside the vagrant box to generate the traces if you face difficulty in installing it from the source code.
- The evaluation on the 'cdschecker-benchmarks' suffice. You may generate a larger trace by changing the input.
- Some applications (e.g. firefox) require significantly more computation and memory. You may skip these.

Restriction Using the already computed clock vectors (CVs) from C11Tester (or any other tool) traces is not admissible.

References

[1] Mark Batty, Scott Owens, Susmit Sarkar, Peter Sewell, and Tjark Weber. Mathematizing C++ concurrency. In *POPL'11*, pages 55–66. ACM, 2011.

[2] Peter Alvaro and Kyle Kingsbury. Elle: Inferring isolation anomalies from experimental observations. *Proc. VLDB Endow.*, 14(3):268–280, 2020.

[3] Weiyu Luo and Brian Demsky. *C11Tester: A Race Detector for C/C++ Atomics*, page 630–646. 2021.

[4] Weiyu Luo and Brian Demsky. C11tester artifact.