Profiling for Miri

Practical Work Proposal

Supervisor: Max Vistrup

Abstract

Miri is an interpreter for Rust. It executes the program strictly according to the rules of the formal abstract semantics of Rust. As such, it can detect most kinds of Rust undefined behavior (UB), and is usually used to test whether a certain piece of Rust code has UB. Miri works by simulating the Rust program step by step, checking all the UB rules at each step. For example, it checks whether values are well-formed or whether data races are happening right now.

Unfortunately, this makes Miri quite slow. Some of this slowdown is expected: Since Miri does a bunch of extra work for each step of the simulated program, it is expected that it takes much longer to evaluate a complete program. But since Miri is already quite slow, performance regressions often go unnoticed. Also, various features might be implemented in less-than-optimal ways, further causing unnecessary slowdowns.

Goals

Before Miri can be made faster, it is necessary to figure out which parts of Miri are the main bottleneck. So, the goal of this project is to extend Miri with support for profiling and performance tracing, and to determine the main bottlenecks.

This is not as easy as it sounds: On the one hand, Miri has to do work to check all the rules of the Rust abstract machine. On the other hand, Miri also has to run the actual code, which might just be slow. Of course, this is not a clean separation: Some code is slow because it uses features that then in turn make Miri do more work to check all the abstract machine/UB rules. The specific goal is to find ways of measuring which category (or what overlap of categories) is responsible for the high execution time, and also nice ways of visualizing this.

Some possible tools that could be useful for this are tracing-chrome, a crate that can use "tracing" infrastructure (which already somewhat exists in Miri, but would need to be extended/adjusted) to produce files that can then be viewed with the Chrome trace viewer, and tracy – but this is not a full survey; part of the project would involve determining the right tool for this task.

This profiling and measuring on its own would already be a huge improvement, since it lays the foundation for later optimizing some of the slow parts.

Optional Goals

Actually making Miri faster is not a required goal of the thesis. This is because we are not really sure how much potential for optimization is there, due to the missing profiling. But this does not mean that you are forbidden from optimizing Miri. Such optimization attemps should however be guided by profiling data, instead of being ad-hoc.

Requirements

- Knowledge of Rust is required

- A good grasp of computer performance, asymptotic vs. actual runtimes, how the cache or profiling works, ... is advantageous. (Un)fortunately, there is no single course we can require here. The following courses seem to teach some of it, so having done well in any is advantageous:
  - 252-0028-00L: Digital Design and Computer Architecture and 252-0026-00L: Algorithms and Data Structures should cover the basics
  - 263-0007-00L: Advanced Systems Lab goes into detail
  - 263-0006-00L: Algorithms Lab should leave you with a good understanding of peformance, as well
  - Having experience with Competitive Programming can help as well

Further Reading

In 2023, Ralf gave a talk about Miri and undefined behavior, which was recorded. For more on undefined behavior, see this for an explainer targeting C programmers (but many of the same principles apply to Rust).