Partial specialization for subtype polymorphism in Virgil

Yonah Goldberg (ygoldber) **Jason Yao (jlyao)**

Project URL: https://yonahg.com/optcomp

Project Milestone

Major Changes

There are not many major changes, we just decided to write specialization code for standalone methods instead of class constructors since we deemed those easier, but we still plan to write the other if we have enough time.

We will also need to write other optimizations that can take advantage of this optimization, see the Surprises section for more info.

Accomplishments

We have successfully specialized methods to more specific subtypes of their arguments, which allows previously virtual method calls on an argument to be replaced with non-virtual calls on the exact class. We also replaced the corresponding calls to the original method with calls to the specialized methods. Currently, we blindly specialize to all subtypes of all function parameters, with no heuristics.

We have a draft PR of the changes we've made so far here: https://github.com/titzer/virgil/pull/296

Meeting the Milestone

We did complete the initial partial specialization of methods (our proposal specified class constructors instead) and wrote some micro benchmarks to determine the effectiveness of our optimization. Unfortunately, we could not observe any meaningful differences in runtime. This is likely due to the virtual method calls themselves not dominating the runtime of the overall programs, and we would likely see an improvement after performing more optimizations enabled by devirtualizing method calls.

We had hoped that we could simply run optimizations to clean up code after specialization. For example, once you specialize, you can eliminate type queries and inline virtual method calls. After inlining, you might have the opportunity for even more optimizations. We were able to turn virtual method calls into non-virtual method calls. However, inlining in Virgil is currently only supported via an `SsaEarlyInliner`, which inlines by regenerating SSA from the AST, and so you can not run it in the middle of the compiler during subtype specialization.

Surprises

We wrote some micro-benchmarks for subtype specialization. To our surprise, they didn't show any measurable performance benefits. One contrived benchmark had a millions of virtual dispatch eliminations and still no performance benefit. We did however see a measurable effect on number of instructions executed. We think there is not a large benefit because the virtual dispatch eliminations in a loop are all to the same address, so the processor can play tricks with ILP and caching. We think that we can achieve a larger benefit with inlining the de-virtualized calls and applying other optimizations after inlining.

Logistics

Revised Schedule

Week	Goals
11/18	**Jason:** • Start writing inlining and other optimizations to be run after specialization **Yonah:** • Begin implementing heuristics for deciding when to perform specialization
11/25	**Jason:** • Finish post-specialization optimizations • If we have time, explore implementing class constructor specialization. **Yonah:** • Finish heuristics. • Collect results from benchmarks with heuristics.
12/2	**Both:** • Collect graphs and tables and finish final report.

Resources Needed

We have all the resources we need already, and they have not changed from the project proposal.