Partial specialization for subtype polymorphism in Virgil

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

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

Project Description

Introduction

Virgil is a statically typed, object-oriented style language, with some functional features. We wish to implement partial specialization, where we can generate multiple instances of a polymorphic class/function that expect a more specific type than the one given by the programmer.

Motivation

Because Virgil contains object-oriented paradigms, there are many instances where subtype polymorphism occurs. Also, since the Virgil compiler optimizes a whole program, the compiler can access much more specific information at every use of subtype polymorphism.

This allows for more effective optimizations to specialize classes/functions that have polymorphic members/parameters compared to compilers that have separate compilation units. Specialization not only eliminates virtual dispatch, but also allows for aggressive inlining after the fact, which can have a drastic affect on performance, especially in some common use cases.

Example Optimization Opportunity

In the visitor design pattern for object-oriented programming, a class has an immutable and polymorphic visitor field, which has functionality for visiting different parts of a data structure. This design pattern is often used in interpreters. For example, in visiting an AST, you might call a `visit_add` or `visit_sub` instruction, etc… When using the visitor, you pattern match on the part of the data structure and then call the appropriate visit method.

By specializing the class for the specific type of visitor, you can eliminate all the virtual dispatch calls to the visitor, which often occur in a loop. Then you can inline the visiting methods, leading to even more performance gains.

Goals

We first aim to apply partial specialization to immutable class members, and then possibly if we have time to partial function applications. The difficulty in this optimization will be figuring out when we want to apply specialization, because there is a tradeoff between code size and speedup. We will decide to use an explicit annotation and/or heuristics to decide.

75% Goal

Implementation of the partial specialization code for class constructors.

100% Goal

Heuristics to balance runtime and code size for the class specialization.

125% Goal

Partial specialization for class constructors and function calls.

Logistics

Schedule

Week	Goals

10/21	**Both team members:**
	• Complete proposal
	• Conduct more research on partial specialization.
	  ▸ When are good cases to specialize other than the visitor design pattern?
	  ▸ Class specialization vs. function specialization
	  ▸ Assemble list of relevant research papers
	• Collect existing examples of Virgil code that can benefit from this optimization

10/28	**Both team members:** • Familiarize ourselves with the Virgil compiler infrastructure.
11/4	• **Jason**: Begin writing code to collect specialization candidates (for now, class constructor calls with a subtype of the constructor parameter). • **Yonah**: Begin writing code to specialize a candidate.
11/11	• **Jason**: Complete rough implementation of class specialization. • **Yonah**: Write and evaluate some micro benchmarks.
11/18	**Both team members (unsure how to split at this time):** • Finish implementation of class specialization. • Create and evaluate larger, representative benchmarks. • Begin implementing heuristics for deciding when to perform specialization
11/25	**Both team members (unsure how to split at this time):** • Finish heuristics. • Collect results from representative benchmarks with heuristics. • If we have time, explore implementing method/function specialization.
12/2	**Both team members (unsure how to split at this time):** • Collect graphs and tables and finish final report.

Milestone (11/20)

Have a complete implementation of partial specialization for class constructors, with micro benchmark results to gauge the initial effectiveness of the optimization.

Literature Search

- Customization: optimizing compiler technology for SELF, a dynamically-typed object-oriented programming language
- Automatic Program Specialization for Java

We definitely need to look for more papers on this topic.

Resources Needed

We will work on the open source Virgil compiler and likely evaluate our performance optimizations on Wizard, a WASM engine. We don't require any specialized hardware since Virgil targets x86, JVM, and WASM.

We have all the resources we need already.

Getting Started

- We have met with Ben Titzer brainstorming this project topic. We will likely need to meet with him again to get greater aquainted with the Virgil compiler and to refine our ideas.