Advanced Compiler Design

Spring Semester 2016

Project

The final assignment for the course is to implement, test, and evaluate an advanced compiler optimization, language feature, or program analysis for Javali.

1 General Concerns

You may choose to implement an optimization technique or program analysis from the literature or a known feature from another, similar programming language. However, to obtain the highest possible grade, you have to incorporate and experiment with your own ideas. Alternatively, you can come up with some ideas that (to your and our knowledge) have not been implemented before. For example, this could be a specific combination of optimization techniques or a language feature that comes from a different domain or is new altogether. If you plan to pursue a project based primarily on your own ideas, please discuss it with the assistants *before* the project proposal presentation.

You are required to do an *evaluation* of your implementation. For optimizations, your evaluation will naturally be about performance. For language features, this is not necessarily the case. For example, if you implement interfaces, you *may* (and probably should) evaluate the performance of interface calls, but you may also evaluate the usability aspect, e.g., by comparing your design to other languages.

2 Schedule & Deliverables

During the course of the project, you are required to provide various deliverables and presentations to show your progress.

Project Proposal *Due: April 22, 8 am*

The project proposal is an outline of what you plan to achieve by the end of the semester. It has two forms: a written proposal with an extent of *one to three* pages and a *three to five minute* oral presentation. Describe the objective of the project, the basic approach, your own ideas, and how you plan to evaluate your implementation. In addition, specify which parts you plan to complete for the milestone (see below).

On April 22, you will give the presentation in class. Submit a first version of the written proposal until 8 am on the same day (i.e., before the presentations). If you have any doubts or questions before that, contact the assistants. After the presentations, they will give you feedback and may ask you to adapt your proposal. *Submit the updated proposal until April 29*.

Milestone Presentation *Due: May 18*

Three weeks after you have submitted your final project proposal, you present your first results. If you implement an optimization, an analysis, or a feature from the literature, the core parts of your implementation should be there. This allows you to spend the last two to three weeks for experimenting with your own ideas and, equally important, for the evaluation. The presentation is oral and should again take *three to five minutes*.

Final Presentation *Due: June 1*

At the end of the semester, you give a final presentation of *five to eight minutes*, focusing on the work of the last two weeks. Present the results of the evaluation, the impact of your own ideas, and your conclusions.

Project Report and Code *Due: June 10*

After the presentation, you have one more week to finish your project by submitting a written report and the source code of your implementation. The report should be phrased as a scientific essay and should include some details about your implementation and evaluation. The report must not exceed *six pages*. Use the template provided on the course website.

Your implementation should (obviously) be correct and thoroughly tested. Unit tests, test programs, and any testing or evaluation infrastructure are part of the submission. Further, we expect the code to be well-structured and documented.

3 Project Examples

Here are some examples of possible projects. You are welcome to select a project not in this list, either from the literature or based on your own ideas. Note that the listed examples can (and for the most part should) be extended and combined with your own ideas.

Loop optimizations

The purpose of this project is to investigate loop optimizations. Possible optimizations include:

- Loop unrolling, loop peeling
- Loop reversal
- Loop splitting
- Loop fusion/loop fission
- Loop interchange
- Loop invariant code motion
- Loop unswitching

You may want to add your own, restricted loop construct to Javali, e.g., a `foreach` loop.

Method calls optimization

Method resolution optimizations in the context of object oriented systems have been of interest to various research groups. The main idea behind these optimizations is to avoid the indirection necessary for resolving the method to be invoked for an object.

- Inline caching
- Method inlining
- Procedure cloning/specialization

Remember that you probably need to perform some analysis (e.g. profiling, class hierarchy analysis) to apply these techniques.

Object layout optimizations

In this project, you investigate the interaction between the CPU architecture and the memory layout of objects. You will explore different strategies for representing objects in memory that defer from a simple linear layout of the fields. For example, you can think about fusing or splitting objects based on their access patterns.

Parallelism in Javali

The goal of this project is to introduce parallelism constructs to Javali. Since the basic approach using explicit `Thread`s and `synchronized` constructs is very simple, think about constructs like Futures/Promises, parallel loops or streams.

Exceptions for Javali

In this project you will implement exceptions and exception handlers (like `try-catch`). You may copy the syntax and semantics from Java or come up with your own design. Choices you will have to make include:

- How to support predefined and user-defined exception classes
- Whether to support *checked* (i.e. declared) exceptions
- How to implement exceptions in the code generation backend

Interfaces and traits

Javali doesn't have interfaces. In this project you first implement interfaces inside the framework. This means introducing a new kind of declaration type. The compiler then must perform semantic checks to see if all the interface methods are implemented in the class or in one of its super classes.

You may also implement support for Java-8-style *default methods* in interfaces or even Scala-style *traits*.

Multiple inheritance

Multiple inheritance is a popular feature that introduces many interesting problems to solve. In this project you implement multiple inheritance in the framework. Different programming languages have implemented multiple inheritance in different ways, for example, see the approaches taken by Python or C++. Try to get inspired by existing solutions, explore what are the implementation consequences from a compiler design point of view.

Type inference

Many modern statically typed programming languages (e.g., Scala) feature some flavour of type inference, which frees the programmer from having to explicitly declare the types for variables, parameters, fields, etc. Instead, the compiler infers the type for each variable, based on the assignments to this variable. The inferred type is then used to type check the program. In this project you design and implement a type inference algorithm for the Javali language. In addition, you come up with a suitable syntax to declare a variable's type to be inferred.

Value types for Javali

In contrast to normal classes, value types are user defined types with semantics similar to primitive types. Instances of value types may consist of multiple parts (like fields), but they do not have an identity. Value types are (also) interesting from a optimization point of view, because often they can be implemented more efficiently than classes.

Advanced program analysis

Traditionally a compiler's task is to verify that the program complies to the language specification and to optimize the program's performance. However, the infrastructure that the compiler provides can also be used for advanced program analysis, for example, to detect potential programming errors (i.e., bugs).

One such example is to detect whether the programmer accidentally passed method arguments with the same type in the wrong order, using compile-time information like variable names. Another possibility is to insert profiling instructions during code generation in order to analyze the runtime behavior of programs.

If you are interested in such a project, please refer to the assistants who will direct you to past and current research, or come up with your own analysis.