4190.570 Advanced Compiler Construction Fall 2018

Term Project: Building an Optimizing Compiler for SnuPL/1

The goal of this term project is to implement, test, and evaluate an advanced compiler optimization into an existing compiler.

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 to our knowledge) have not been implemented before. For example, this could be a specific combination of optimization techniques that comes from a different domain or is new altogether. You are required to do an evaluation of your implementation in terms of performance of the optimized code. Do not forget to include other relevant metrics such as code size if relevant for your optimization

2. Schedule & Deliverables

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

Project Proposal – September 19, 2018. 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). Please stick to these limits (number of pages, length of presentation).
You will present your project proposal in class on the 19th of September. After the presentation, you may receive feedback and be asked you to adapt your proposal. The final version must be submitted by October 1, 2018. The final version will be graded (and contributes towards your project score).

Milestone Presentation – October 29, 2018. Four weeks after you have submitted your final project proposal, you will present your first results. An analysis or a feature from the literature, and the core parts of your implementation should be there. This allows you to spend the last six weeks for completing the implementation, experimenting with your own ideas and evaluating your project. The presentation will be oral and should take 3-5 minutes.

Final Presentation – December 10+12, 2018. At the end of the semester, you will give a final presentation of five to eight minutes. Present your implementation, the results of the evaluation, the impact of your own ideas, and your conclusions.

Project Report and Code – December 19, 2018. After the final presentation, you have one week to finalize your code, including test cases, evaluations, and results. You will also write and submit a report that should be phrased as a scientific essay and should include some details about your implementation and evaluation. The report must not exceed eight pages of the template that will be provided on the course website.
Your implementation should be correct and thoroughly tested. Unit tests, test programs, and any testing or evaluation infrastructure are part of the submission. We expect your code to be well-structured and documented.

3. SnuPL/1 Compiler

The SnuPL/1 compiler is a compiler developed as part of the undergraduate compiler course (4190.409). It is a procedural language and has only three basic data types (integer, character, and boolean) and one composite data type (array). It currently compiles to IA32 (32-bit x86) code only.

You will be given access to the compiler framework comprising pre-compiled object files and source code. All optimizations should be performed on the low-level IR (three address code) of the compiler even though, in principle, optimizations on the abstract syntax tree are also possible.

The intermediate representation and the backend are provided in source code form. You are free to modify them as you see fit.

4. Optimization Examples

This section presents some examples of possible projects. You are welcome to choose a project which is not on this list, either from literature or based on your own ideas. Note that the listed examples are incomplete ideas, i.e., they will not give you the full grade for this course. They must be extended and combined with your own ideas. The complete ideas should be proposed and presented by you with the project proposal.

4.1 Register allocation

Implement a register allocator for SnuPL/1. You are free to output either IA32 or x86-64 assembly code. Obviously, x86-64 offers more room for optimizations.
Register allocation can be implemented using the traditional graph coloring approach, linear scan, or an SSA-based algorithm.
Literature:

- textbook
- A. Appel: “Modern Compiler Implementation in Java/C”

4.2 Constant propagation and dead code elimination

For this project, you implement constant propagation based on data-flow analysis. You will need to develop a data-flow analysis pass. This can be extended to interprocedural constant propagation or function call optimization (see below).
Literature:

- textbook

4.3 Partial-redundancy elimination

Implement partial-redundancy elimination based on traditional data-flow analysis or SSA. You will need to develop a data-flow analysis pass.
Literature:

- textbook

4.4 Function call optimizations

Several optimization techniques to speed-up function calls are possible. Function inlining or function specialization (i.e., generating different code depending on incoming parameter values) are ideas that fit into this category.
Literature:

- textbook
- Cooper et al., A Methodlogy for Procedure Cloning, Computer Languages, Volume 19, 1993

4.5 Function memoization

Memoization is a technique that aims at reducing execution time by storing and retrieving the results of repeated function invocations with the same input.

Literature:

- Pugh et al., Incremental Computation via Function Caching, POPL ’89
- Ding et al., A Compiler Scheme for Reusing Intermediate Computation Results, CGO ’14
- Suresh et al., Intercepting functions for memoization, TACO ’15
- Suresh et al., Compile-Time Function Memoization, CC ’17

4.6 Your own ideas

You are welcome to propose and implement your own optimization for snuplc.

5. Logistics

You will use Subversion (SVN) as a code repository for your development and also as a means to submit your deliverables.

Each team will receive an individual SVN repository at

```
https://snupl.snu.ac.kr/svn/<teamname>
```

and individual logins for the repository.

Happy coding!