Final Paper

Overview

This document expands on the high-level description of the paper requirements from the syllabus.

The paper should be submitted as a `pdf` file.

Compiler Overview

The paper should begin with an overview of the architecture of your compiler and the important data structures used within. In particular, discuss the different phases of your compiler (listed below). Each of the following can be addressed in a paragraph or two.

- Parsing: How does your compiler parse a given input file? How is the input file represented within your compiler (i.e., data structure is used internally)?
- Static Semantics: Discuss how your compiler implements static type checking and the check for a return along each path through a function.
- Intermediate Representation: Discuss the representation of the program as a set of control flow graphs with LLVM instructions. What purpose does the control flow graph serve? Why does a compiler use an intermediate representation like LLVM that is more abstract than actual assembly instructions? What is the benefit of SSA-form?
- Optimizations: State the specific optimizations implemented. Discuss the idea of abstract interpretation/evaluation of instructions as relates to constant propagation.
- Code Generation and Register Allocation: Make note of any interesting aspects of your translation from LLVM to assembly. Why must the phi instructions be eliminated and how does your compiler do this?
- Other: Discuss any other aspect of your particular implementation that you believe is interesting. This is a great opportunity to share something about which you are proud.

Analysis

This is your opportunity to present and consider some analysis of the output of your compiler.

For each benchmark (for which your compiler generates valid output), compare the following configurations. If your compiler is capable of generating actual assembly, then compare the runtime of the code generated by your compiler against the installed compiler on the target machine (e.g., gcc). If your compiler generates LLVM, but not assembly, then compare against clang.

Include a statement about where these experiments were performed. Provide a graph for each benchmark comparing the time for each configuration. If you cannot execute a configuration, then make note of that in the graph. If your compiler can generate both stack-based LLVM and register-based LLVM, then include a table comparing the instruction count (only LLVM instructions) for each version for each benchmark (this, of course, hides the instructions that might be added if register allocation were performed).

Configurations

- Stack-based IR (translated to assembly).
- Register-based IR (translated to assembly).
- With optimizations enabled (for whichever IR your compiler supports for optimizations).
- gcc/clang on provided .c file without optimizations (-O0).
- gcc/clang on provided .c file with optimizations (-O3).