Superoptimization for LLVM IR

Rafael Auler
rafaelauler@gmail.com
IC-UNICAMP, Brazil

April 6, 2011

1 Objective

This project focuses on implementing superoptimization algorithms targeted at the LLVM IR. The project uses arbitrary LLVM bitcode as a training set to discover new peephole optimizations that can be later integrated into LLVM instruction simplify phase or as a separate peephole optimization pass.

2 Motivation

Code optimization is the task of using a faster sequence of instructions which compute the same information, compared to a previous code sequence. The old sequence may then be substituted by the new one, creating a faster program with the same behavior. The problem of finding a code sequence that does some computation in optimal time is undecidable and, in practice, optimizations are discovered by humans using analytical reasoning. Finding new optimizations in an automated fashion, using a search algorithm, is a different approach to the problem and became known in literature as superoptimization, term coined by Massalin [2].

Using computer-aided optimization discovery, different kind of optimizations unnoticed by humans may be detected, specially those specific to the target machine code [1]. This project will focus on implementing a superoptimizer which will exhaustively investigate new optimizations for the LLVM IR. Using this approach, I hope to greatly contribute to the current LLVM instruction simplify phase, introducing new substitutions identified in this project.

3 Methodology

There is already a superoptimizer for LLVM in initial stages of development. The existing superoptimizer currently limits its search space to a few transformations. This project will focus on improving this current superoptimizer by

extending the number of transformations applied and thus greatly increasing the search space.

In order to test the superoptimizer efficiency, it will use LLVM bitcodes compiled from the entire LLVM test suite and SPEC to harvest possible code sequence optimizations. When successfully identified, these optimizations could be implemented directly as peephole optimizations for LLVM IR.

4 Timeline

This project is expected to last three months, as follows.

- **1st month, 1st week** - Study the LLVM IR in order to establish a list of axioms, that is, valid transformations that determine the search algorithm capability. There is an important trade-off to investigate here, since using fewer transformations will lead to faster searches, but several good optimizations may be missed.
- **1st month, 2nd week** - Thorough study of the current implemented base and new algorithm design to support arbitrary transformations.
- **1st month, 3rd week until end of project** - Implementation of the search algorithm.
- **2nd month** - Test phase begins. Test algorithm with simple transformations searching for optimized code sequences.
- **3rd month** - Harvesting new optimizations and implementing fixes to the search algorithm as more performance feedback from training data is received.

5 Biography

I am a Master student at the University of Campinas (UNICAMP), Brazil (timezone GMT -3). In my master thesis, I have already worked with superoptimization theory and automatic code sequence searching. But instead of using it for optimization purposes, I use this to automatically generate a LLVM backend based on a machine description using the ArchC architecture description language. The backend patterns are fixed and the implementation of the pattern is inferred using a search algorithm similar to the one used on superoptimizers. Thus, I have experience both on superoptimizer theory and LLVM code.

References

[1] Sorav Bansal and Alex Aiken. Automatic generation of peephole superoptimizers. *SIGPLAN Not.*, 41:394–403, October 2006.

[2] Henry Massalin. Superoptimizer: a look at the smallest program. In *ASPLOS-II: Proceedings of the second international conference on Architectual support for programming languages and operating systems*, pages 122–126, Los Alamitos, CA, USA, 1987. IEEE Computer Society Press.