KT-2

Advanced Compiler Design

Introduction to Optimizing Compilers, Virtual Machines and Runtime Systems

Administrivia

- Instructor:
  - Kostis Sagonas (MIC, Hus 1, 352)
- Course home page
  http://user.it.uu.se/~kostis/Teaching/KT2-12/
- If you want to be enrolled in the course, you have to send, by January 20th, a mail with your name and your UU e-mail address to:
  kostis@it.uu.se

Course Goals

- Familiarize yourself with the theoretical basis of advanced compiler optimizations
- Give you a general orientation on the map of compiler optimization techniques
- Give you a general understanding of
  - how some modern programming language features and constructs are implemented
  - the tradeoffs that are involved in including some feature in a modern programming language or not

Course Goals and Requirements

Non-Goals:

- Overview all possible compiler optimizations
- Cover compilation techniques for parallelism/multicores

Requirements:

- You are supposed to be familiar with basic programming language implementation concepts
  - In particular, with semantic analysis and code generation
  - However, these topics will not be needed in this course!
- You are supposed to know how to program in a high-level language (esp. in some functional language)

Course Content

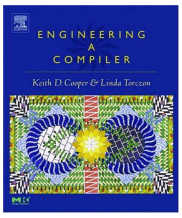
- Static analysis and optimization
  - Theory for Static Analysis
  - Optimization Algorithms
- Implementation techniques for high-level languages
  - Memory Management (aka Garbage Collection)
  - Virtual Machines & Bytecode Interpreters
  - Just-in-time (JIT) Compilers
  - Feedback-Directed Compilation

Course Structure

- Course has theoretical and practical aspects
  - Need both in modern optimizing compilers!
- Lectures get you up-to-date with various topics and the state-of-the-art in programming language implementation.
- Project (can be done in groups of 2 or 3)
  - get you exposed with the real issues that need to be addressed when implementing a compiler optimization
  - teach you how to plan the development and testing of a non-trivial piece of software
  - teach you how to perform a serious performance evaluation.

Course's Literature

Slides of lectures posted on the web



- In addition to lecture slides, various papers from the recent research on programming language design and implementation will be available at the course's homepage
- These handouts are **required** reading

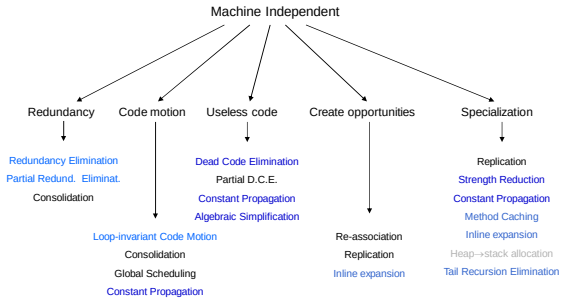
Course Syllabus (Tentative)

- Introduction to advanced compiler design
- Using static analysis for global optimization
- Foundations of static analysis and abstract interpretation
- Static Single Assignment (SSA): Construction and Use
- Global register allocation
- Automatic memory management
- Virtual machines and interpretation techniques
  - Just-in-time (JIT) compilers
  - Dynamic and feedback-directed compilation
- Implementation of Object Oriented Languages
- Implementation of Garbage Collectors for Java

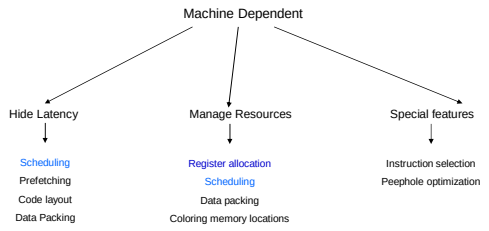
Introduction to Compiler Optimization

- The most important aspect of a compiler optimization is that the **program remains correct**
- The terminology is confusing and misleading:
  - Global means function-local
  - Optimization means improvement
  - Compilation time vs. runtime speedup is often a factor to take into account
- The next slides try to give you a taxonomy of some common compiler optimization techniques

Taxonomy of Global Compiler Optimizations



Taxonomy of Global Compiler Optimizations



Why is this course interesting?

- Optimization is a very challenging problem —you can not write an ideal compiler: there is always room for improvements
- The course will teach you many techniques and tools that you can use in other areas
- You will gain a better understanding of how a compiler works and what to expect of the code generated by compilers
- You will learn how to work with a code base which is too big for a single person to grasp and you learn how to do test-based development and perf. evaluation
- It is fun!