Compiling Techniques

Lecture 1: Introduction

Christophe Dubach

22 September 2015

Table of contents

Essential Facts

- Lecturer: Christophe Dubach (christophe.dubach@ed.ac.uk)
- Office hours: Thursdays 11am-12pm
- Textbook (not strictly required):
  - Keith Cooper & Linda Torczon: Engineering a Compiler Elsevier, 2004
  - Textbook can be reused in UG4 Compiler Optimisation
- Course website:
  `http://www.inf.ed.ac.uk/teaching/courses/ct/`
- Discussion forum:
  `https://piazza.com/#winter2016/infr10053`

Action

Create an account and subscribe to the course on piazza.

- Evaluation: no exam, coursework only

Coursework

- Write a full compiler for a subset of C
  - Will be written in Java
  - Backend will target Java bytecode
  - Compiled code executable in a Java Virtual Machine (JVM)
- Four parts:
  - week 4 (10pts) Parser
  - week 6 (20pts) Abstract Syntax Tree (AST) builder
  - week 8 (20pts) Semantic analyser
  - week 11 (50pts) Code generator
- Course work is hard!
  - Last year about 20% of students failed the coursework!

Coursework continued

- Automated system to evaluate coursework
  - Mark is a function of how many programs compile successfully
  - Nightly build of your code with scoreboard
- Will rely on git/bitbucket

Action

Create an account on `http://bitbucket.org` and send account id to daniel.hillerstrom@ed.ac.uk with subject "CT course bitbucket account id"

- Tutorials here to help with coursework
  - Friday 15:10 - 16:00, Forrest Hill, Room 3.D02 (tutors: Christophe Dubach, Bjoern Franke)

Class-taking Technique

- Extensive use of projected material
  - Attendance and interaction encouraged
- Reading book is optional
- Not a programming course!
- Start the practical early

Syllabus

- Overview
- Scanning
- Parsing
- Abstract Syntax Tree
- Semantic analysis
- Code generation
  - Java ByteCode
  - Code shapes
- Advanced topics
  - Instruction selection
  - register allocation

Compilers

What is a compiler?

A program that *translates* an executable program in one language into an executable program in another language.
The compiler might improve the program, in some way.

What is an interpreter?

A program that directly *execute* an executable program, producing the results of executing that program

Examples:

- C is typically compiled
- R is typically interpreted
- Java is compiled to bytecode, then interpreted or compiled (just-in-time) within a Java Virtual Machine (JVM)

A Broader View

Compiler technology = Off-line processing

- Goals: improved performance and language usability
- Making it practical to use the full power of the language
- Trade-off: preprocessing time versus execution time (or space)
- Rule: performance of both compiler and application must be acceptable to the end user

Examples:

- Macro expansion / Preprocessing
- Database query optimisation
- Javascript just-in-time compilation
- Emulation acceleration: TransMeta code morphing

Why study compilation?

- Compilers are important system software components:
  they are intimately interconnected with architecture, systems, programming methodology, and language design
- Compilers include many applications of theory to practice:
  scanning, parsing, static analysis, instruction selection
- Many practical applications have embedded languages:
  commands, macros, formatting tags
- Many applications have input formats that look like languages:
  Matlab, Mathematica
- Writing a compiler exposes practical algorithmic & engineering issues:
  approximating hard problems; efficiency & scalability

Intrinsic interest

Compiler construction involves ideas from many different parts of computer science

Artificial intelligence	Greedy algorithms Heuristic search techniques
Algorithms	Graph algorithms Dynamic programming
Theory	DFA & PDA, pattern matching Fixed-point algorithms
Systems	Allocation & naming Synchronisation, locality
Architecture	Pipeline & memory hierarchy management Instruction set
Software engineering	Design pattern (visitor) Code organisation

Intrinsic merit

Compiler construction poses challenging and interesting problems:

- Compilers must do a lot but also run fast
- Compilers have primary responsibility for run-time performance
- Compilers are responsible for making it acceptable to use the full power of the programming language
- Computer architects perpetually create new challenges for the compiler by building more complex machines
- Compilers must hide that complexity from the programmer
- Success requires mastery of complex interactions

Making languages usable

It was our belief that if FORTRAN, during its first months, were to translate any reasonable "scientific" source program into an object program only half as fast as its hand coded counterpart, then acceptance of our system would be in serious danger.

...

I believe that had we failed to produce efficient programs, the widespread use of languages like FORTRAN would have been seriously delayed.

John Backus

Next lecture

The View from 35000 Feet

- How a compiler works
- What I think is important
- What is hard and what is easy