263 - 28100 Advanced Compiler Design
Thomas Gross

Lecture
Wed 10-12
Fr 9-10

Lab
Fr 10-12

Recitation

May swap Lab sessions & lectures this Friday: lec 9-11

www.1st.inf.ethz.ch/teaching

- announcements
- copies of slides
- assignments
Credit:
- Assignments (3 planned)
  - build/extend a compiler
    - [Java, framework]
    - optimizing compiler infrastructure
  50% of your grade

- Project - build on assignments
  50% of your grade

Teams of 2
- default: equal credit to both partners
- 1-1 meetings are possible
- different allocation if necessary

If there are problems, special situations:
- send me mail
- or contact TA (Luca Delio Toffolo)
tentative dates

Apr 24: project plan presentation
May 17: status update
May 31st: final presentation
May 31st: final report due (6 pages...)

→ 1 week extension o.k.
Advanced Compiler Design

Core topic: optimization techniques for modern programming languages

Optimization: key issue for real compiler techniques have many uses
Optimization: not necessarily optimal (cannot prove)

Better than naive translation

Program transformation
Program analysis

In addition: compiler engineering
- effectiveness (when?) efficient? (how well does the compiler work?)
1.0 Program Representation

IR: intermediate representation

a) Source program → IR
   IR must capture relevant details
   \( \leftrightarrow \) influence on transformation

b) Object code → IR
   - Binary translators
     - rewriter
     - translate from
       - PPC → X86
     - insert checks
     - pass postpass optimizers / link time optimization
Review: (Basic) Compiler Design

IR: forest of trees

\[ x = a + b \]  \( \Rightarrow \) 1
\[ d = x + 1 \]  \( \Rightarrow \) 2
\[ b = a + c \]  \( \Rightarrow \) 3
\[ i f ( \cdots ) d \cdots 3 \]  \( \Rightarrow \) 4
\[ y = b + x \]  \( \Rightarrow \) 5

- Simple
- One stmt at a time can be translated

not a good IR for transformations or optimizations
This IR causes problems for an optimizing compiler

- No reuse of values
  (x is computed by stmt 1, used by stmt 2; hidden in the graph)

- All values read from or written to memory
  (stmt 2 reads x, but x was written by stmt 1)

- Over specification
  (order of stmts is fixed: first stmt 1, then stmt 3, then stmt 5...)

details & dependencies (which stmt uses which value(s)) are implicit!
1.1. Basics

For an optimizing compiler we need an IR that:

- makes data dependences explicit
- captures producer-consumer relationship
- makes control dependences explicit
  - determines order of execution
  (sequencing)

Ideally, the IR contains only constraints derived from the source program.
  - no superfluous constraints
control dependences

We use **basic blocks as nodes in the Control Flow Graph (CFG)**.

A **basic block (bb)** is a maximal sequence of statements that are always executed together.  

- Assumption: no exception is generated either from machine (assembly) instructions, source statements, byte codes, ...

Either all statements in a **bb** are executed or none.
Example
( High level language program )

```plaintext
if (... )
  t
else
  x. proc()
  for (i = 0; i < N; i++)
    i
```

10
Example

(RISC asm)

```
id x, R1
id y, R2
add R1, R2, R3
beq R3, L1
```

```
add R5, R6, R7
id z, R6
add...
```

```
L1:
id
id
```

There might be a protection violation.

```
if R3 == 0 then goto L1
```
- data dependences

```
1. x = a + b
2. d = x + 1
3. b = a + c
4. i += ...
5. y = b + x
```

**Wish list**
- recognize producer - consumer relationships
- identify reuse of operands
- make sure "b" in
  - stmt 1 is read before
  - stmt 3 writes b.

One safe solution: stmt 1 is executed before stmt 3 (sufficient)

What really matters is: stmt 1 reads the old value and
stmt 3 writes the new value.