Compiler Construction

BITS Pilani
Pilani Campus

Vinti Agarwal
March 2021

innovate achieve lead

BITS Pilani
Pilani Campus

CS F363, Compiler Construction

Lecture topics: Optimization Overview

Optimization Overview

- optimization is second last compiler phase
- most complexity in modern compiler is in the optimizer
- also by far the largest phase

Optimization Overview

When should we perform optimization?

- on AST

  AST

  IR

  pro: machine independent

  con: too high level

- On assembly language

  pro: expose optimization opportunities

  con: machine dependent

  con: must reimplement optimizations when retargeting

- On intermediate language

  pro: machine independent

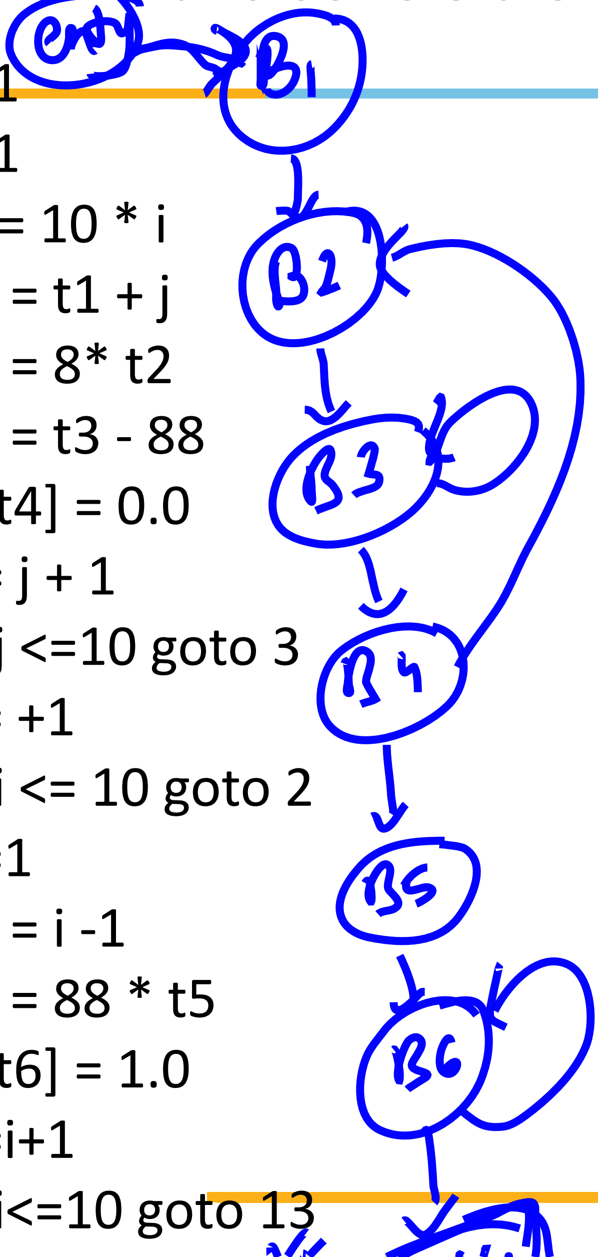
  pro: expose optimization opportunities

Intermediate code example

```
1)   i=1
2)   j=1
3)   t1= 10 * i
4)   t2 = t1 + j
5)   t3 = 8* t2
6)   t4 = t3 - 88
7)   a[t4] = 0.0
8)   j = j + 1 ✓
9)   if j <=10 goto 3
10)  i = +1
11)  if i <= 10 goto 2
12)  i =1
13)  t5 = i -1
14)  t6 = 88 * t5
15)  a[t6] = 1.0
16)  i =i+1
17)  if i<=10 goto 13
```

Intermediate code example

```
1)  i=1
2)  j=1
3)  t1= 10 * i
4)  t2 = t1 + j
5)  t3 = 8* t2
6)  t4 = t3 - 88
7)  a[t4] = 0.0
8)  j = j + 1
9)  if j <=10 goto 3
10) i = +1
11) if i <= 10 goto 2
12) i =1
13) t5 = i -1
14) t6 = 88 * t5
15) a[t6] = 1.0
16) i =i+1
17) if i<=10 goto 13
```



Turns a 10 x 10 matrix into an identity matrix

?

```
for $i$ from 1 to 10 do
        for $j$ from 1 to 10 do
                $a[i,j] = 0.0;$
for $i$ from 1 to 10 do
        $a[i,i] = 1.0;$
```

B1 → 1

B2 → 2

B3 → 3 - 9

B4 → 10 - 11

B5 → 12

exit

Basic blocks

- A basic block is a maximal sequence of instructions with:
  - no labels (except at first instruction)
  - no jumps except at the last instruction
- idea:
  - cannot jump into a basic block (except at beginning)
  - cannot jump out of a basic block (except at the end)
  - a basic block is a single entry, single exit, straight line code segment

Basic blocks example

1) L:
2) t: = 2*x
3) w:= t + x
4) if w >0 goto L

valid block ✓

(3) executes only after (2) ✓

we can change (3) to w: = 3*x

can we eliminate (2) as well

Basic blocks

- How to know when a basic block begins and ends?
- Leaders: the first instruction of a basic block. ✓
- Rules:
  - ✓ First instruction of three address code
  - ✓ Any instruction target of a conditional or unconditional jump
  - ✓ any instruction immediately follows a a conditional or unconditional jump

Control Flow graph

- A directed graph with
  - basic blocks as nodes
  - An edge from block A to block B if execution passes from the last instruction in A to the first instruction in B
    - e.g. last instruction in A is jump LB
    - execution can fall through from block A to block B.

Optimization seeks to improve a program resource utilization

- execution time (most often) ✓
- code size ✓
- disk access ✓
- memory usage ✓

Optimization ✓ should not alter what the program computes ✓

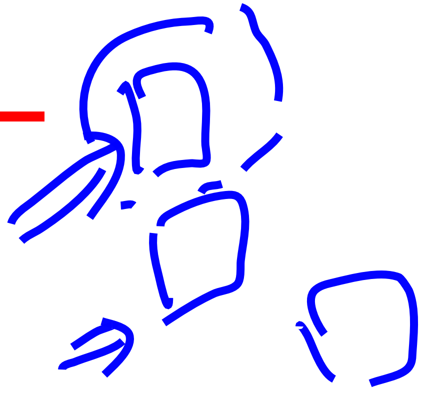
- The answers must still be the same

✓

Granularities of optimizations

1. Local optimizations
   - apply to a basic block in isolation
2. Global optimization
   - apply to a control flow graph (method body) in isolation
3. inter-procedural optimizations
   - apply across methods boundaries

Most compiler do (1), many do (2), few do (3)



Granularities of optimizations

- In practice, often a conscious decision is made not to implement the fanciest optimization know
- why?
  - some optimization are hard to implement
  - some optimization are costly in compilation time
  - some optimizations have low payoff
  - many fancy optimization are all three

Goal: maximum benefit for minimum cost

Local Optimization

- The simplest form of optimization ✓

- Optimize one basic block
  - No need to analyze whole procedure body

Local Optimization

- Some statement can be deleted

  x := x + 0

  x := x * 1

- Some statements can be simplified

  x := x *0 → x: = 0

  y := y **2 → y: = y * y

  x := x * 8 → x:=x<<3

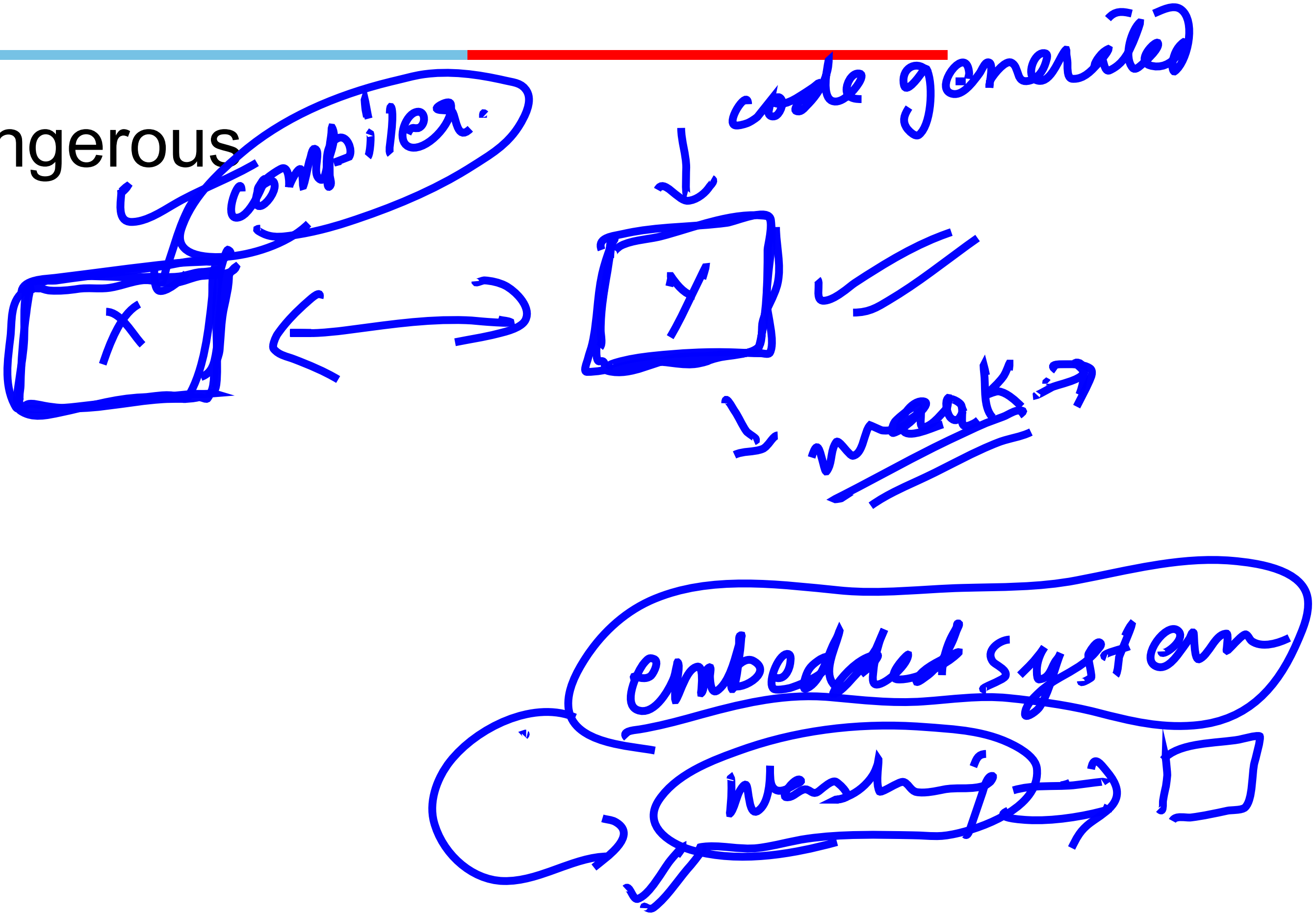
  x := x *15 → t:=x<<4; x := t - x

Constant Folding

- Operations on constants van be computed at compile time
    - statement x := y op z
    - y and z are constants
    - then y and z can be computed at compile time
- Examples
    - x := $\overbrace{2 + 2}$ ✓ ⇒ x:=4 ✓
    - ✓ if 2 < 0 jump L ⇒ deleted
    - ✓ if $\boxed{2>0}$ jump L ⇒ jump L

Constant Folding

- can be dangerous



Thank You!