Introduction to Optimization

Kristoffer H. Rose
krisrose@cs.nyu.edu

Compiler Construction
CSCI-GA.2130-001/Fall 2013
NYU Courant Institute

December 5, 2013

1ALSU §§8.5,7
Context

source program → Front End → Code Optimizer → Code Generator → target program

Introduction to Optimization

Kris Rose
Basic Block DAG

Peephole Optimization

Thanks!

HACS Q&A

Introduction to Optimization

Kris Rose
DAG – Directed Acyclic Graph

- One node per initial value.
- One node per statement, with an edge to last node observing every parameter.
- Node labeled by operator and list of variables *not* used further.
- Output nodes are those with live exit variables.
DAG uses

- Local common subexpressions.
- Dead code elimination.
- Apply algebraic simplifications.
- Reorder statements to reduce variable count.
Local Common Subexpressions

1. \( a = b + c \)
2. \( b = a - d \)
3. \( c = b + c \)
4. \( d = a - d \)
Local Common Subexpressions

1. a = b + c
2. b = a - d
3. c = b + c
4. d = a - d

\[
\begin{align*}
+ & \quad + \quad + \\
\text{c} & \quad \text{b}, \text{d} & \quad \text{d}_0 \\
- & \quad + \\
\text{b}_0 & \quad \text{a} & \quad \text{c}_0
\end{align*}
\]
Dead Code Elimination

**Inputs:** $b, c, d$

1. $a = b + c$
2. $b = b - c$
3. $c = c + d$
4. $e = b + c$

**Outputs:** $a, b$
Dead Code Elimination

Inputs: \( b, c, d \)

1. \( a = b + c \)
2. \( b = b - c \)
3. \( c = c + d \)
4. \( e = b + c \)

Outputs: \( a, b \)
Algebraic Identities

\[
x + 0 = 0 + x = x \\
x \times 1 = 1 \times x = x \\
2 \times x = x + x \\
x/2 = x \times 0.5
\]
Algebraic Identities

\[ x + 0 = 0 + x = x \quad \text{\textbf{EXPENSIVE}} \]
\[ x - 0 = x \quad \text{\textbf{CHEAPER}} \]
\[ x \times 1 = 1 \times x = x \]
\[ x/1 = x \]

\[ x^2 = x \times x \]
\[ 2 \times x = x + x \]
\[ x/2 = x \times 0.5 \]
Algebraic Identities

- **Constant folding**
- **Commutativity** with Local Common Subexpressions
- **Associativity** with composite expressions
Algebraic Identities

- **Constant folding**
- **Commutativity** with Local Common Subexpressions
- **Associativity** with composite expressions

1. \( a = c + b \)
2. \( e = c + d + b \)
Algebraic Identities

- Constant folding
- **Commutativity** with Local Common Subexpressions
- **Associativity** with composite expressions

1. \( a = c + b \)
2. \( e = c + d + b \)
Introduction to Optimization

Array References

1. \( x = a[i] \)
2. \( a[j] = y \)
3. \( z = a[i] \)
Basic Block DAG

Peephole Optimization

Introduction to Optimization

Kris Rose

Array References

1. \( x = a[i] \)
2. \( a[j] = y \)
3. \( z = a[i] \)

![Diagram showing array references and basic block DAG]

Killed!
No Pointing!

1. \( x = \ast p \)
2. \( \ast q = y \)

Needs pointer ("points-to") analysis!
No Pointing!

1. \( x = *p \)
2. \( *q = y \)

Needs pointer ("points-to") analysis!
1. \[ a = b + c \]
2. \[ b = a - d \]
3. \[ c = b + c \]
4. \[ d = b \]

Respect ordering, pay special attention to overlapping side effects!
a = b + c
b = a - d
c = b + c
d = b

Respect ordering, pay special attention to overlapping side effects!
Respect ordering, pay special attention to overlapping side effects!
Introduction to Optimization

1. Basic Block DAG
2. Peephole Optimization
3. Thanks!
4. HACS Q&A
Redundant Load&Store

1. LD R0, a
2. ST a, R0

1. LD R0, a
Redundant Load&Store

LD R0, a
ST a, R0

↓

LD R0, a
Unreachable Code

if debug == 1 goto L1
  goto L2
L1: compute-and-print-really-big-expensive-stuff
L2:

if debug != 1 goto L2
L1: compute-and-print-really-big-expensive-stuff
L2:

With debug = 0,
L2:
Unreachable Code

if debug == 1 goto L1
goto L2
L1: compute-and-print-really-big-expensive-stuff
L2:

↓

if debug != 1 goto L2
L1: compute-and-print-really-big-expensive-stuff
L2:

With debug = 0,
L2:
Unreachable Code

1. if debug == 1 goto L1
2. goto L2
3. L1: compute-and-print-really-big-expensive-stuff
4. L2:

↓

1. if debug != 1 goto L2
2. L1: compute-and-print-really-big-expensive-stuff
3. L2:

With debug = 0,

1. L2:
Flow of Control

1. goto L1
2. ...
3. L1: goto L2

1. goto L2
2. ...
3. L1: goto L2
Flow of Control

```
1   goto L1
2   ...
3  L1: goto L2
```

↓

```
1   goto L2
2   ...
3  L1: goto L2
```
Flow of Control

1. goto L1
2. ...
3. L1: if a < b goto L2
4. L3:

↓

1. if a < b goto L2
2. goto L3
3. ...
4. L3:
Flow of Control

1. `goto L1`

2. ...

3. L1: `if a < b goto L2`

4. L3:

↓

1. `if a < b goto L2`

2. `goto L3`

3. ...

4. L3:
Just the Beginning...

- Peephole vs Good Code Generator
- Everything above has better analysis-based solutions
1. Basic Block DAG
2. Peephole Optimization
3. Thanks!
4. HACS Q&A
Finally . . .

- Evaluations!
- Project Milestone 3 on Sunday.
- Exam:
  - 5:10 pm–7:00 pm, December 12, 2013, Room WWH 312.
  - Closed book.
  - All ALSU sections quoted on class web page.
1. Basic Block DAG

2. Peephole Optimization

3. Thanks!

4. HACS Q&A
The End