Classical Optimizations

Overview

- Last lecture
  - Code generation
  - High-level languages
  - Optimization
  - Goals, approaches

- This lecture
  - Classical optimizations
    - Common subexpression elimination
    - Forward substitution / copy propagation
    - Loop invariant code motion
    - Strength reduction
    - Loop test elimination, induction variable elimination
    - Constant propagation
    - Dead code elimination
  - Basic block construction
    - Basic block DAG construction

Example Code

Fortran Source Code:

```fortran
sum = 0
do 10 i = 1, n
  sum = sum + a(i) * a(i)
10  sum = 0
```

3-address code

```fortran
1. sum = 0
2. i = 1
3. if i >> n goto 15
4. t1 = addr(a) - 4
5. t2 = i * 4
6. t3 = t1[t2]
7. t4 = addr(a) - 4
8. t5 = i * 4
9. t6 = t4[t5]
10. t7 = t3 * t6
11. t8 = sum + t7
12. sum = t8
13. i = i + 1
14. goto 3
15. sum = 0
```

Control Flow Graph (CFG)

Common Subexpression Elimination

```fortran
1. sum = 0
2. i = 1
3. if i >> n goto 15
4. t1 = addr(a) - 4
5. t2 = i * 4
6. t3 = t1[t2]
7. t4 = addr(a) - 4
8. t5 = i * 4
9. t6 = t4[t5]
10. t7 = t3 * t6
11. t8 = sum + t7
12. sum = t8
13. i = i + 1
14. goto 3
15. sum = 0
```
Forward Substitution (Copy Propagation)

1. sum = 0
2. i = 1
3. if i > n goto 15
4. t1 = addr(a) - 4
5. t2 = i * 4
6. t3 = t1[t2]
10a. t7 = t3 * t3
12a. sum = sum + t7
13. i = i + 1
14. goto 3
15.

Invariant Code Motion

1. sum = 0
2. i = 1
2a. t1 = addr(a) - 4
2b. t2 = i * 4
5. t2 = t2 + 4
6. t3 = t1[t2]
10a. t7 = t3 * t3
12a. sum = sum + t7
13. i = i + 1
14. goto 3
15.

Strength Reduction

1. sum = 0
2. i = 1
2a. t1 = addr(a) - 4
2b. t2 = i * 4
2c. t9 = n * 4
3. if t2 > t9 goto 15
6. t3 = t1[t2]
10a. t7 = t3 * t3
12a. sum = sum + t7
12b. t2 = t2 + 4
13. i = i + 1
14. goto 3
15.

Loop Test Adjustment

1. sum = 0
2. i = 1
2a. t1 = addr(a) - 4
2b. t2 = i * 4
2c. t9 = n * 4
3. if t2 > t9 goto 15
6. t3 = t1[t2]
10a. t7 = t3 * t3
12a. sum = sum + t7
12b. t2 = t2 + 4
13. i = i + 1
14. goto 3a
15.

Induction Variable Elimination

1. sum = 0
2. i = 1
2a. t1 = addr(a) - 4
2b. t2 = i * 4
2c. t9 = n * 4
3a. if t2 > t9 goto 15
6. t3 = t1[t2]
10a. t7 = t3 * t3
12a. sum = sum + t7
12b. t2 = t2 + 4
13. i = i + 1
14. goto 3a
15.

Constant Propagation

1. sum = 0
2. i = 1
2a. t1 = addr(a) - 4
2b. t2 = i * 4
2c. t9 = n * 4
3a. if t2 > t9 goto 15
6. t3 = t1[t2]
10a. t7 = t3 * t3
12a. sum = sum + t7
12b. t2 = t2 + 4
14. goto 3a
15.
Dead Code Elimination

1. \text{sum} = 0
2. \text{t1} = \text{addr}(a) - 4
2a. \text{t2} = \text{i} * 4
3a. \text{if t2 > t9 goto 15}
6. \text{t3} = \text{t1[t2]}
10a. \text{t7} = \text{t3} * \text{t3}
12a. \text{sum} = \text{sum} + \text{t7}
14. \text{goto 3a}
15.

Final Optimized Code

1. \text{sum} = 0
2. \text{t1} = \text{addr}(a) - 4
3. \text{t2} = 4
4. \text{t4} = \text{i} * 4
5. \text{if t2 > t4 goto 11}
6. \text{t5} = \text{t1[t2]}
7. \text{t5} = \text{t3} * \text{t3}
8. \text{sum} = \text{sum} + \text{t5}
9. \text{t2} = \text{t2} + 4
10. \text{goto 5}
11.

unoptimized: 8 temps, 11 stmts in innermost loop
optimized: 5 temps, 5 stmts in innermost loop

1 index addressing                    2 index addressing
1 multiplication                     3 multiplications
2 additions                          2 additions & 2 subtractions
1 jump                               1 jump
1 test                               1 test
1 copy

CFG of Final Optimized Code

Basic Block Construction

• Find leader statements
  • First program statement
  • Targets of conditional or unconditional gotos
  • Any statement following a conditional goto
  • \((\forall x \in \text{leaders})\) construct Bx, basic block headed by x: include all statements following x until next leader or end of program is reached.
  • At end of algorithm, any statements not in some basic block are unreachable from program entry and are therefore, dead code.

3-address code example

1. \text{sum} = 0
2. \text{i} = 1
3. \text{if i} > \text{a} \text{goto 15}
4. \text{t1} = \text{addr}(a) - 4
5. \text{t2} = \text{i} * 4
6. \text{t3} = \text{t1[t2]}
7. \text{t5} = \text{i} * 4
9. \text{t6} = \text{t4[t5]}
10. \text{t7} = \text{t3} * \text{t6}
11. \text{t8} = \text{sum} + \text{t7}
12. \text{sum} = \text{t8}
13. \text{i} = \text{i} + 1
14. \text{goto 3}
15.
3-address code example

<table>
<thead>
<tr>
<th>Basic Block</th>
<th>Leader</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. sum = 0</td>
<td></td>
</tr>
<tr>
<td>2. i = 1</td>
<td></td>
</tr>
<tr>
<td>3. if i &gt;&gt; 4 goto 15</td>
<td></td>
</tr>
<tr>
<td>4. t1 = addr(a) + 4</td>
<td></td>
</tr>
<tr>
<td>5. t2 = i + 4</td>
<td></td>
</tr>
<tr>
<td>6. t3 = t1[t2]</td>
<td></td>
</tr>
<tr>
<td>7. t4 = addr(a) - 4</td>
<td></td>
</tr>
<tr>
<td>8. t5 = i + 4</td>
<td></td>
</tr>
<tr>
<td>9. t6 = t4[t5]</td>
<td></td>
</tr>
<tr>
<td>10. t7 = t3 + t6</td>
<td></td>
</tr>
<tr>
<td>11. t8 = sum + t7</td>
<td></td>
</tr>
<tr>
<td>12. sum = t8</td>
<td></td>
</tr>
<tr>
<td>13. i = i + 1</td>
<td></td>
</tr>
<tr>
<td>14. goto 3</td>
<td></td>
</tr>
</tbody>
</table>

Common Subexpression Elimination

- **Common subexpression (CSE)**
  - Portion of expressions repeated multiple times
  - If computes same value, can reuse previously computed value

- **Directed acyclic graph (DAG)**
  - Program representation that exposes CSEs
  - Unlike tree, nodes can have multiple parents
  - No cycles allowed

- **Building expression DAGs**
  - Maintain hash table for leaves, expressions
  - Reuse nodes found in hash table when building DAG

Directed Acyclic Graph Example

- Consider the following expression
  - \( a + c \times (b - c) \times (b - c) \times d \)

  **Representations**
  - Tree
  - Directed acyclic graph

Directed Acyclic Graphs

- **What about assignment?**
  - Identical subexpressions may have different values
  - Must ensure each value has unique node in DAG

- **Renaming**
  - Add subscripts to variables
  - Increment subscript for LHS after assignment
  - Variable references use new subscript

- **Example**
  - Original
  - After renaming
    - \( a = a + b \)
    - \( c = a + b \)
    - \( d = a + d \)

Directed Acyclic Graphs – Renaming Example

- Notes
  - LHS of assignment becomes label for node
  - Nodes may have multiple labels if expression is reused

Basic Block DAG Construction

Reference: ASU, p. 548
Basic Block DAG Construction

How to add a subexpression into a partially constructed DAG:

Is there a node already for B + C? node(B) + node(C) defined?
- If so, add A to its list of labels.
- If not:
  - Is there a node labeled B already? node(B) defined?
    - If not, create a leaf labeled B.
  - Is there a node labeled C already? node(C) defined?
    - If not, create a leaf labeled C.
  - Create a node labeled A, for +, with left child B and right child C.

How to do this? HASHING op, node(opd1), node(opd2)

Reference: ASU, p.548