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
  10 sum = sum + a(i) * a(i)
```

3-address code

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

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. `T`

---

**Common Subexpression Elimination**

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`
10a. `t7 = t3 * t3`
11. `t8 = sum + t7`
12. `sum = t8`
13. `i = i + 1`
14. `goto 3`
15. `common subexpression elimination`
Forward Substitution (Copy Propagation)

1. \( \text{sum} = 0 \)
2. \( i = 1 \)
3. if \( i > n \) goto 15
4. \( t1 = \text{addr}(a) - 4 \)
5. \( t2 = i \times 4 \)
6. \( t3 = t1[t2] \)
10a. \( t7 = t3 \times t3 \)
11. \( t8 = \text{sum} + t7 \)
12. \( \text{sum} = t8 \)
12a. \( \text{sum} = \text{sum} + t7 \)
13. \( i = i + 1 \)
14. goto 3
15.

Invariant Code Motion

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

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

**Loop Test Adjustment**

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

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

Final Optimized Code

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

1. sum = 0
2. t1 = addr[a] - 4
3. t2 = 4
4. t4 = 4 * n
5. if t2 > t4 goto 11
6. t3 = t1[t2]
7. t5 = t3 * t3
8. sum = sum + t5
9. t2 = t2 + 4
10. goto 5

Basic Block Construction

- Find leader statements
  - First program statement
  - Targets of conditional or unconditional gotos
  - Any statement following a conditional goto
- (∀x ∈ 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. `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.
3-address code example

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. goto 3

Basic Block DAG Construction

4. t1 = addr(a) - 4
5. t2 = i * 4
6. t3 = t1[t2]
...

Reference: ASU, p.548
4. \( t_1 = \text{addr}(a) - 4 \)
5. \( t_2 = i \times 4 \)
6. \( t_3 = t_1[t_2] \)
7. \( t_4 = \text{addr}(a) - 4 \)
8. \( t_5 = i \times 4 \)
9. \( t_6 = t_4[t_5] \)
10. \( t_7 = t_3 \times t_6 \)

...
Basic Block DAG Construction

How to add a subexpression into a partially constructed DAG:

\[ A = B + C \]

Is there a node already for \( B + C \)?
- If so, add \( A \) to its list of labels.
- If not:
  - is there a node labeled \( B \) already?
    If not, create a leaf labeled \( B \).
  - Is there a node labeled \( C \) already?
    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