Lecture topics
I Intro to optimizations
II Constant folding
III Data flow analysis
IV Obstacles to optimization
V Intro to optimizations

Big picture

High-level IR \rightarrow \text{Portable optimizer}

Code generator

Low-level IR \rightarrow \text{Machine-dependent optimizer}

Printer

Target code

Typical optimizations

<table>
<thead>
<tr>
<th>Name</th>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant folding</td>
<td>$\text{h}=60 \times 60$</td>
<td>$\text{h}=3600$</td>
</tr>
<tr>
<td>Copy propagation</td>
<td>$\text{t}=\text{h}$</td>
<td>$\text{t}=\text{h}$</td>
</tr>
<tr>
<td>Dead code elimination</td>
<td>$\text{d}=24 \times \text{t}$</td>
<td>$\text{d}=24 \times \text{h}$</td>
</tr>
<tr>
<td>Common subexpression elimination</td>
<td>$\text{h}=60 \times 60$</td>
<td>$\text{h}=60 \times 60$</td>
</tr>
<tr>
<td>Common subexpression elimination</td>
<td>$\text{d}=24 \times 60 \times 60$</td>
<td>$\text{d}=24 \times \text{h}$</td>
</tr>
<tr>
<td>Algebraic simplification</td>
<td>while $\text{i} &lt; \text{n}-1$</td>
<td>while $\text{i} &lt; \text{n}$</td>
</tr>
<tr>
<td>Strength reduction</td>
<td>$\text{x}=\text{y} \times 2$</td>
<td>$\text{x}=\text{y} + \text{y}$</td>
</tr>
<tr>
<td>Strength reduction</td>
<td>$\text{z}=\text{y} / 4$</td>
<td>$\text{z}=\text{y} / 2$</td>
</tr>
<tr>
<td>Loop invariant code motion</td>
<td>while $\text{i} &lt; \text{n}$</td>
<td>while $\text{i} &lt; \text{n}$</td>
</tr>
</tbody>
</table>

Basic block
- sequence of straight-line instructions
- no internal jumps or jump targets
- vertex in control-flow graph

Control-flow graph

Origins of optimization opportunities
- available redundancy in source code (e.g., $24 \times 60 \times 60$ more readable)
- inherent redundancy in source code (e.g., array bounds checks in Java)
- simple compiler pass (e.g., spurious temporaries/copies)
- interactions between optimizations
### III. Data-flow analysis

#### Forward analysis algorithm

- for all blocks $B_i$, initialize $B_i$.in / $B_i$.out
- worklist, put (ENTRY)
- while worklist not empty ?
  - $B = $ worklist, pop()
  - $B$.in = merge $P$.out from predecessors
  - apply transfer functions for instructions in $B$ to compute $B$.out
  - if $B$.out changed ?
    - add successors of $B$ to worklist

#### Constant folding specification

<table>
<thead>
<tr>
<th>Direction</th>
<th>Forwards</th>
</tr>
</thead>
<tbody>
<tr>
<td>Info</td>
<td>For each variable, NC / ? / constant</td>
</tr>
<tr>
<td>Initialization</td>
<td>Parameters NC, rest ?</td>
</tr>
</tbody>
</table>

#### Transfer function:

- Case $x = y + z$:
  - if both $y$ / $z$ are constant:
    - $x$.info = $y$.info + $z$.info
  - else if one of $x$, $y$, $z$.info is NC:
    - $x$.info = NC

#### Merge:

- If $x$.info same on all predecessors
  - $x$.info = that value
  - else:
    - $x$.info = NC

### IV. Obstacles to optimization

#### Uses of data-flow analysis

- optimization (e.g., dead-code elimination)
- register allocation (e.g., use liveness for lifetime graph in graph coloring)
- bug-finding (e.g., uninitialized variable)

#### IV.1. Calls

- Before
  - while $i < p - i$, $j = p = p + (2^i)$
  - while $i < p$

- After
  - $x = p - f$
  - $q, f = 5$
  - return $p, f$
  - return $x$

#### IV.2. Only legal if...

- $q$ unchanged
- pow side-effect free
- pow deterministic

#### IV.3. Solutions

- inlining
- interprocedural analysis

#### IV.4. Pointers

- Before
  - $x = p - f$
  - $q, f = 5$
  - return $p, f$
  - return $x$

- After
  - $x = p - f$
  - $q, f = 5$
  - return $p, f$
  - return $x$

#### IV.5. Only legal if...

- $p$ and $q$ point to different records

#### IV.6. Solutions

- may-alias analysis
- scalar replacement

### Reminders

- Fri 12/12: pr 4 due
- Wed 12/7: last lecture
- Fri 12/9: hw8 due
- Fri 12/16: pr 5 due
- Wed 12/21: final exam