1.8 Application: Loop-invariant Code

Example

\[
\text{for } (i = 0; i < n; i++)
\]
\[
a[i] = b + 3;
\]

// The expression \( b + 3 \) is recomputed in every iteration.
// This should be avoided!

The Control-flow Graph

Idea Transform into a do-while-loop...
\[ \text{while}(e) \quad \Rightarrow \quad \text{if}(e) \{ \text{do} \quad \text{while} \quad \text{end} \} \]

... now there is a place for \( T = e_1 \).

Idea
Transform into a do-while-loop ...

Application of T5 (PRE):

\[
\begin{array}{c|c|c}
A & B \\
\hline
0 & \emptyset & \emptyset \\
1 & \emptyset & \emptyset \\
2 & \emptyset & \{b + 3\} \\
3 & \{b + 3\} & \emptyset \\
4 & \{b + 3\} & \emptyset \\
5 & \{b + 3\} & \emptyset \\
6 & \{b + 3\} & \emptyset \\
7 & \emptyset & \emptyset
\end{array}
\]
Application of **T5** (PRE):

**Conclusion**

- Elimination of partial redundancies may move loop-invariant code out of the loop.
- This only works properly for do-while-loops!
- To optimize other loops, we transform them into do-while-loops before-hand:

\[
\begin{align*}
&\text{while } (b) \text{ strat} \\
&\quad \text{do strat} \\
&\quad \text{while } (b); \\
&\quad \Rightarrow \text{Loop Rotation}
\end{align*}
\]

**Problem**

If we do not have the source program at hand, we must re-construct potential loop headers

\[
\Rightarrow \text{ Pre-dominators}
\]

\[
u \text{ pre-dominates } v, \text{ if every path } \pi : \text{start} \rightarrow^* v \text{ contains } u.
\]

We write: \( u \Rightarrow v. \)

\( \Rightarrow \) is reflexive, transitive and anti-symmetric.

**Caveat** \( T = b + 3; \) may not be placed before the loop:

\[
\Rightarrow \text{ There is no decent place for } T = b + 3;.
\]
Conclusion

- Elimination of partial redundancies may move loop-invariant code out of the loop.
- This only works properly for do-while-loops!
- To optimize other loops, we transform them into do-while-loops beforehand.

```
while (b) stmt  if (b)
do  stmt
while (b);
```

Loop Rotation

Problem

If we do not have the source program at hand, we must re-construct potential loop headers

\[ \rightarrow \] \text{Pre-dominators}

\( u \text{ pre-dominates } v \), if every path \( \pi : \text{start} \rightarrow^* v \) contains \( u \).
We write: \( u \Rightarrow v \).

\( \Rightarrow \) is reflexive, transitive and anti-symmetric.

Computation

We collect the nodes along paths by means of the analysis:

\[
P = 2^{Nodes}, \quad \subseteq = \supseteq \quad \llbracket (\_ , \_ , v) \rrbracket P = P \cup \{v\}
\]

Then the set \( P[v] \) of pre-dominators is given by:

\[
P[v] = \bigcap \{ \llbracket \pi \rrbracket P \mid \pi : \text{start} \rightarrow^* v \}
\]

Example

<table>
<thead>
<tr>
<th></th>
<th>( P )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>{0}</td>
</tr>
<tr>
<td>1</td>
<td>{0, 1}</td>
</tr>
<tr>
<td>2</td>
<td>{0, 1, 2}</td>
</tr>
<tr>
<td>3</td>
<td>{0, 1, 2, 3}</td>
</tr>
<tr>
<td>4</td>
<td>{0, 1, 2, 3, 4}</td>
</tr>
<tr>
<td>5</td>
<td>{0, 1, 5}</td>
</tr>
</tbody>
</table>
Since \( \mathcal{H} \) are distributive, the \( P[v] \) can computed by means of fixpoint iteration ...

Example

<table>
<thead>
<tr>
<th></th>
<th>( P )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>{0}</td>
</tr>
<tr>
<td>1</td>
<td>{0, 1}</td>
</tr>
<tr>
<td>2</td>
<td>{0, 1, 2}</td>
</tr>
<tr>
<td>3</td>
<td>{0, 1, 2, 3}</td>
</tr>
<tr>
<td>4</td>
<td>{0, 1, 2, 3, 4}</td>
</tr>
<tr>
<td>5</td>
<td>{0, 1, 5}</td>
</tr>
</tbody>
</table>

The partial ordering \( \rightarrow \) in the example:

<table>
<thead>
<tr>
<th></th>
<th>( P )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>{0}</td>
</tr>
<tr>
<td>1</td>
<td>{0, 1}</td>
</tr>
<tr>
<td>2</td>
<td>{0, 1, 2}</td>
</tr>
<tr>
<td>3</td>
<td>{0, 1, 2, 3}</td>
</tr>
<tr>
<td>4</td>
<td>{0, 1, 2, 3, 4}</td>
</tr>
<tr>
<td>5</td>
<td>{0, 1, 5}</td>
</tr>
</tbody>
</table>

Apparently, the result is a tree.
In fact, we have:

**Theorem**
Every node \( v \) has at most one immediate pre-dominator.

**Proof**
Assume:
there are \( u_1 \neq u_2 \) which immediately pre-dominate \( v \).
If \( u_1 \Rightarrow u_2 \) then \( u_1 \) not immediate.
Consequently, \( u_1, u_2 \) are incomparable.
Now for every $\pi : \text{start} \rightarrow^* v$:

$$\pi = \pi_1 \pi_2 \quad \text{with} \quad \pi_1 : \text{start} \rightarrow^* u_1$$

$$\pi_2 : u_1 \rightarrow^* v$$

If, however, $u_1, u_2$ are incomparable, then there is path:

$\text{start} \rightarrow^* v \quad \text{avoiding} \quad u_2$:

---

Observation

The loop head of a while-loop pre-dominates every node in the body.

A back edge from the exit $u$ to the loop head $v$ can be identified through

$$v \in \mathcal{P}[u]$$

Accordingly, we define:

$$\bigcup \quad \text{predominates} \quad \bigcap$$

---

... in the Example

```
\begin{verbatim}
0
1
2
3
4
5
6
7

i = 0;
y = b + 3;
A_i = A + 1;
M[A_i] = 1;
i = i + 1;
\end{verbatim}
```

Neg(i < n)  Pos(i < n)

Neg(\text{not } i < n)  Pos(\text{not } i < n)
In the Example

Caveat

There are unusual loops which cannot be rotated:

... but also common ones which cannot be rotated:

Here, the complete block between back edge and conditional jump should be duplicated.
... but also common ones which cannot be rotated:

Here, the complete block between back edge and conditional jump should be duplicated.

1.9 Eliminating Partially Dead Code

Example

\[ T = x + 1; \]

\[ M[x] = T; \]

\[ x + 1 \] need only be computed along one path.

Idea

\[ T = x + 1; \]

\[ M[x] = T; \]
Problem

- The definition \( x = e \) (\( x \not\in \text{Vars}_e \)) may only be moved to an edge where \( e \) is safe.
- The definition must still be available for uses of \( x \).

We define an analysis which maximally delays computations:

\[
\begin{align*}
\llbracket \llbracket x = e \rrbracket \rrbracket D &= D \\
\llbracket x = e \rrbracket D &= \begin{cases} 
D \setminus (\text{Use}_e \cup \text{Def}_e) \cup \{ x = e \} & \text{if } x \not\in \text{Vars}_e \\
D \setminus (\text{Use}_e \cup \text{Def}_e) & \text{if } x \in \text{Vars}_e
\end{cases}
\end{align*}
\]

Problem

- The definition \( x = e \) (\( x \not\in \text{Vars}_e \)) may only be moved to an edge where \( e \) is safe.
- The definition must still be available for uses of \( x \).

We define an analysis which maximally delays computations:

\[
\begin{align*}
\llbracket \llbracket x = e \rrbracket \rrbracket D &= D \\
\llbracket x = e \rrbracket D &= \begin{cases} 
D \setminus (\text{Use}_e \cup \text{Def}_e) \cup \{ x = e \} & \text{if } x \not\in \text{Vars}_e \\
D \setminus (\text{Use}_e \cup \text{Def}_e) & \text{if } x \in \text{Vars}_e
\end{cases}
\end{align*}
\]

... where:

\[
\begin{align*}
\text{Use}_e &= \{ y = e' \mid y \in \text{Vars}_e \} \\
\text{Def}_x &= \{ y = e' \mid y = x \wedge x \in \text{Vars}_e \}
\end{align*}
\]

For the remaining edges, we define:

\[
\begin{align*}
\llbracket x = M[e] \rrbracket D &= D \setminus (\text{Use}_e \cup \text{Def}_e) \\
\llbracket M[e_1] = e_2 \rrbracket D &= D \setminus (\text{Use}_e \cup \text{Use}_e) \\
\llbracket \text{Pos}(e) \rrbracket D &= \llbracket \text{Neg}(e) \rrbracket D = D \setminus \text{Use}_e
\end{align*}
\]
Problem

- The definition $x = c; \ (x \notin Vars_e)$ may only be moved to an edge where $e$ is safe.
- The definition must still be available for uses of $x$.

$$\implies \mathcal{D} = \{ D \} \subseteq \mathcal{D}$$

We define an analysis which maximally delays computations:

$$\begin{align*}
\tensordelay D & = D \\
\tensordelay [x = c] D & = \begin{cases} 
D \setminus (Use_e \cup Def_e) \cup \{ x = c \} & \text{if } x \notin Vars_e \\
D \setminus (Use_e \cup Def_e) & \text{if } x \in Vars_e
\end{cases}
\end{align*}$$

We conclude:

- The partial ordering of the lattice for delayability is given by $\sqsubseteq$.
- At program start: $D_0 = \emptyset$.

Therefore, the sets $D[u]$ of at least delayable assignments can be computed by solving a system of constraints.

- We delay only assignments $a$ where $aa$ has the same effect as $a$ alone.
- The extra insertions render the original assignments as assignments to dead variables ...

Caveat

We may move $y = c;$ beyond a join only if $y = c;$ can be delayed along all joining edges:

Here, $T = x + 1;$ cannot be moved beyond $1$ !!!

Transformation 7

$$\mathcal{D} \setminus \{ a \} = \{ a_1, a_2 \}$$

$$\begin{align*}
\mathcal{D}(\bar{v}) & = \{ a_1 \} \\
\mathcal{D}(\bar{v}) & = \{ a_1 \}
\end{align*}$$
Transformation 7

\[ a \in \mathcal{D}(\sigma) \setminus \{\text{pos}\}(\mathcal{D}(\sigma)) \]
\[ a \in \mathcal{L}(\sigma) \setminus \{\text{pos}\}(\mathcal{D}(\sigma)) \]

Remark

Transformation \( T_7 \) is only meaningful, if we subsequently eliminate assignments to dead variables by means of transformation \( T_2 \).

In the example, the partially dead code is eliminated:

\( T = x + 1; \)

\[ \mathcal{D} \]

\| 0 \| \| 1 \| \| 2 \| \| 3 \| \| 4 \|
---|---|---|---|---|---|
| \( \emptyset \) | \( \{ x \} \) | \( \{ x \} \) | \( \emptyset \) | \( \emptyset \) |

\[ \mathcal{L} \]

\| 0 \| \| 1 \| \| 2 \| \| 3 \| \| 4 \|
---|---|---|---|---|---|---|
| \( \{ x \} \) | \( \{ x \} \) | \( \{ x \} \) | \( \emptyset \) | \( \emptyset \) | \( \emptyset \) | \( \emptyset \) |
Remarks

- After $T7$, all original assignments $y = e; \quad y \notin Vars_{e}$ are assignments to dead variables and thus can always be eliminated.
- By this, it can be proven that the transformation is guaranteed to be non-degrading efficiency of the code.
- Similar to the elimination of partial redundancies, the transformation can be repeated.

Conclusion

→ The design of a meaningful optimization is non-trivial.
→ Many transformations are advantageous only in connection with other optimizations !
→ The ordering of applied optimizations matters !!
→ Some optimizations can be iterated !!!
2 Replacing Expensive Operations by Cheaper Ones

2.1 Reduction of Strength

(1) Evaluation of Polynomials

\[ f(x) = a_n \cdot x^n + a_{n-1} \cdot x^{n-1} + \ldots + a_1 \cdot x + a_0 \]

<table>
<thead>
<tr>
<th></th>
<th>Multiplications</th>
<th>Additions</th>
</tr>
</thead>
<tbody>
<tr>
<td>naive</td>
<td>( \frac{1}{2}n(n + 1) )</td>
<td>( n )</td>
</tr>
<tr>
<td>re-use</td>
<td>( 2n - 1 )</td>
<td>( n )</td>
</tr>
<tr>
<td>Horner-Scheme</td>
<td>( n )</td>
<td>( n )</td>
</tr>
</tbody>
</table>

Idea

\[ f(x) = (\ldots ((a_n \cdot x + a_{n-1}) \cdot x + a_{n-2}) \ldots) \cdot x + a_0 \]

(2) Tabulation of a polynomial \( f(x) \) of degree \( n \):

\[ \rightarrow \text{To recompute } f(x) \text{ for every argument } x \text{ is too expensive.} \]

\[ \rightarrow \text{Luckily, the } n\text{-th differences are constant} \]

![Diagram](image-url)
2 Replacing Expensive Operations by Cheaper Ones

2.1 Reduction of Strength

(1) Evaluation of Polynomials

\[
f(x) = a_n \cdot x^n + a_{n-1} \cdot x^{n-1} + \ldots + a_1 \cdot x + a_0
\]

<table>
<thead>
<tr>
<th>Method</th>
<th>Multiplications</th>
<th>Additions</th>
</tr>
</thead>
<tbody>
<tr>
<td>naive</td>
<td>( \frac{1}{2}n(n + 1) )</td>
<td>( n )</td>
</tr>
<tr>
<td>re-use</td>
<td>( 2n - 1 )</td>
<td>( n )</td>
</tr>
<tr>
<td>Horner-Scheme</td>
<td>( n )</td>
<td>( n )</td>
</tr>
</tbody>
</table>

Example:

\[
f(x) = 3x^3 - 5x^2 + 4x + 13
\]

<table>
<thead>
<tr>
<th>( n )</th>
<th>( f(n) )</th>
<th>( \Delta )</th>
<th>( \Delta^2 )</th>
<th>( \Delta^3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>13</td>
<td>2</td>
<td>8</td>
<td>18</td>
</tr>
<tr>
<td>1</td>
<td>15</td>
<td>10</td>
<td>( 26 )</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>25</td>
<td>36</td>
<td>( 80 )</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>61</td>
<td>( 48 )</td>
<td>( 80 )</td>
<td></td>
</tr>
</tbody>
</table>

Here, the \( n \)-th difference is always

\[
\Delta^n(f) = n! \cdot a_n \cdot h^n \quad (h \text{ step width})
\]

\[
f(x) = (\ldots((a_n \cdot x + a_{n-1}) \cdot x + a_{n-2}) \ldots) \cdot x + a_0
\]

(2) Tabulation of a polynomial \( f(x) \) of degree \( n \):

→ To recompute \( f(x) \) for every argument \( x \) is too expensive.

→ Luckily, the \( n \)-th differences are constant !!!