Reuse Optimization

Last time
- Common subexpression elimination (CSE)

Today
- Partial redundancy elimination (PRE)

Partial Redundancy Elimination (PRE)

Partial Redundancy
- An expression (e.g., $x+y$) is partially redundant at node $n$ if some path from the entry node to $n$ evaluates $x+y$, and there are no definitions of $x$ or $y$ between the last evaluation of $x+y$ and $n$

Elimination
- Discover partially redundant expressions
- Convert them to fully redundant expressions
- Remove redundancy

PRE subsumes CSE and loop invariant code motion
**Loop Invariance Example**

**PRE removes loop invariants**
- An invariant expression is partially redundant
- PRE converts this partial redundancy to full redundancy
- PRE removes the redundancy

**Example**

\[
\begin{align*}
1 & \quad x := y \ast z \\
2 & \quad \ldots \\
3 & \quad a := b + c
\end{align*}
\]

\[
\begin{align*}
1 & \quad x := y \ast z \\
2 & \quad \ldots \\
3 & \quad a := b + c
\end{align*}
\]

---

**Implementing PRE [Morel & Renvoise 1979]**

**Big picture**
- Use local properties (availability and anticipability) to determine where redundancy can be created within a basic block
- Use global analysis (data-flow analysis) to discover where partial redundancy can be converted to full redundancy
- Insert code and remove redundant expressions
Local Properties

An expression is locally transparent in block b if its operands are not modified in b.

An expression is locally available in block b if it is computed at least once and its operands are not modified after its last computation in b.

An expression is locally anticipated if it is computed at least once and its operands are not modified before its first evaluation.

Example

\[
\begin{align*}
\text{a} & := \text{b} + \text{c} & \text{Transparent:} & \{\text{b} + \text{c}\} \\
\text{d} & := \text{a} + \text{e} & \text{Available:} & \{\text{b} + \text{c}, \text{a} + \text{e}\} \\
& & \text{Anticipated:} & \{\text{b} + \text{c}\}
\end{align*}
\]

Local Properties (cont)

How are these properties useful?

- They tell us where we can introduce redundancy.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transparent</td>
<td>The expression can be redundantly evaluated anywhere in the block.</td>
</tr>
<tr>
<td>Available</td>
<td>The expression can be redundantly evaluated anywhere after its last evaluation in the block.</td>
</tr>
<tr>
<td>Anticipated</td>
<td>The expression can be redundantly evaluated anywhere before its first evaluation in the block.</td>
</tr>
</tbody>
</table>
Global Availability

**Intuition**
- Global availability is the same as Available Expressions
- If e is globally available at p, then an evaluation at p will create redundancy along all paths leading to p

![Diagram](expr(expr\rightarrow p))

**Flow Functions**
\[
\text{available}_\text{in}[n] = \bigcap_{p \in \text{pred}[n]} \text{available}_\text{out}[p] \\
\text{available}_\text{out}[n] = \text{locally}_\text{available}[n] \cup (\text{available}_\text{in}[n] \cap \text{transparent}[n])
\]

(Global) Partial Availability

**Intuition**
- An expression is partially available if it is available along some path
- If e is partially available at p, then \(\exists\) a path from the entry node to p such that the evaluation of e at p would give the same result as the previous evaluation of e along the path

![Diagram](expr\rightarrow p)

**Flow Functions**
\[
\text{partially available}_\text{in}[n] = \bigcup_{p \in \text{pred}[n]} \text{partially available}_\text{out}[p] \\
\text{partially available}_\text{out}[n] = \text{locally available}[n] \cup (\text{partially available}_\text{in}[n] \cap \text{transparent}[n])
\]
Global Anticipability

Intuition
– If e is globally anticipated at p, then an evaluation of e at p will make the next evaluation of e redundant along all paths from p

Flow Functions
\[
\text{anticipated}_{\text{out}}[n] = \bigcap_{s \in \text{succ}[n]} \text{anticipated}_{\text{in}}[s]
\]
\[
\text{anticipated}_{\text{in}}[n] = \text{locally}_\text{anticipated}[n] \cup \left( \text{anticipated}_{\text{out}}[n] \cap \text{transparent}[n] \right)
\]

Global Possible Placement

Goal
– Convert partial redundancies to full redundancies
– Possible Placement uses a backwards analysis to identify locations where such conversions can take place
  – e ∈ ppin[n] can be placed at entry of n
  – e ∈ ppout[n] can be placed at exit of n

Push Possible Placement backwards as far as possible
Global Possible Placement (cont)

The placement will create a redundancy on every edge out of the block.

**Flow Functions**

\[
ppout[n] = \bigcap_{s \in \text{succ}[n]} ppin[s]
\]

\[
ppin[n] = \text{anticipated_in}[n] \cap \text{partially_available_in}[n] \cap (\text{locally_anticipated}[n] \cup (ppout[n] \cap \text{transparent}[n]))
\]

Will turn partial redundancy into full redundancy.

The middle of the chain.

This block is at the beginning of a chain.

Updating Blocks

**Intuition**

- Perform insertions at top of the chain.
- Perform deletion at the bottom of the chain.

**Functions**

- \( \text{delete}[n] = ppin[n] \cap \text{locally_anticipated}[n] \)
- \( \text{insert}[n] = ppout[n] \cap (\neg ppin[n] \cup \neg \text{transparent}[n]) \cap \neg \text{available_out}[n] \)

Don’t insert it where it’s fully redundant.
Updating Blocks (cont)

Intuition
– Perform insertions at top of the chain
– Perform deletion at the bottom of the chain

Functions
– $\text{delete}[n] = \text{ppin}[n] \cap \text{locally\_anticipated}[n]$
– $\text{insert}[n] = \text{ppout}[n] \cap (\neg \text{ppin}[n] \cup \neg \text{transparent}[n]) \cap \neg \text{available\_out}[n]$

Sandwich Example

<table>
<thead>
<tr>
<th></th>
<th>B0</th>
<th>B1</th>
<th>B2</th>
<th>B3</th>
<th>B4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{transparent}$</td>
<td>$a+b$</td>
<td>$a+b$</td>
<td>$a+b$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\text{locally_available}$</td>
<td>$a+b$</td>
<td>$a+b$</td>
<td>$a+b$</td>
<td>$a+b$</td>
<td></td>
</tr>
<tr>
<td>$\text{locally_anticipated}$</td>
<td>$a+b$</td>
<td>$a+b$</td>
<td>$a+b$</td>
<td>$a+b$</td>
<td></td>
</tr>
<tr>
<td>available_in</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$a+b$</td>
</tr>
<tr>
<td>available_out</td>
<td>$a+b$</td>
<td>$a+b$</td>
<td>$a+b$</td>
<td>$a+b$</td>
<td></td>
</tr>
<tr>
<td>partially_available_in</td>
<td>$a+b$</td>
<td>$a+b$</td>
<td></td>
<td></td>
<td>$a+b$</td>
</tr>
<tr>
<td>partially_available_out</td>
<td>$a+b$</td>
<td>$a+b$</td>
<td>$a+b$</td>
<td>$a+b$</td>
<td></td>
</tr>
<tr>
<td>anticipated_out</td>
<td>$a+b$</td>
<td>$a+b$</td>
<td>$a+b$</td>
<td>$a+b$</td>
<td></td>
</tr>
<tr>
<td>anticipated_in</td>
<td>$a+b$</td>
<td>$a+b$</td>
<td>$a+b$</td>
<td>$a+b$</td>
<td></td>
</tr>
<tr>
<td>$\text{ppout}$</td>
<td>$a+b$</td>
<td>$a+b$</td>
<td>$a+b$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\text{ppin}$</td>
<td>$a+b$</td>
<td></td>
<td>$a+b$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\text{insert}$</td>
<td>$a+b$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\text{delete}$</td>
<td></td>
<td>$a+b$</td>
<td></td>
<td></td>
<td>$a+b$</td>
</tr>
</tbody>
</table>
Example

B1: \( a := b + c \) \quad B2: \( b := b + 1 \) \quad B3: \( a := b + c \)

<table>
<thead>
<tr>
<th></th>
<th>B1</th>
<th>B2</th>
<th>B3</th>
</tr>
</thead>
<tbody>
<tr>
<td>transparent</td>
<td>( b+c )</td>
<td>( b+c )</td>
<td></td>
</tr>
<tr>
<td>locally_available</td>
<td>( b+c )</td>
<td>( b+c )</td>
<td></td>
</tr>
<tr>
<td>locally_anticipated</td>
<td>( b+c )</td>
<td>( b+1 )</td>
<td>( b+c )</td>
</tr>
<tr>
<td>available_in</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>available_out</td>
<td>( b+c )</td>
<td>( b+c )</td>
<td></td>
</tr>
<tr>
<td>partially_available_in</td>
<td>( b+c )</td>
<td>( b+c )</td>
<td></td>
</tr>
<tr>
<td>partially_available_out</td>
<td>( b+c )</td>
<td>( b+c )</td>
<td></td>
</tr>
<tr>
<td>anticipated_out</td>
<td>( b+c )</td>
<td>( b+c )</td>
<td></td>
</tr>
<tr>
<td>anticipated_in</td>
<td>( b+c )</td>
<td>( b+1 )</td>
<td>( b+c )</td>
</tr>
<tr>
<td>ppout</td>
<td>( b+c )</td>
<td>( b+c )</td>
<td></td>
</tr>
<tr>
<td>ppin</td>
<td>( b+c )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>insert</td>
<td></td>
<td>( b+c )</td>
<td></td>
</tr>
<tr>
<td>delete</td>
<td></td>
<td></td>
<td>( b+c )</td>
</tr>
</tbody>
</table>

Comparing Redundancy Elimination

Value numbering
  - Examines values not expressions
  - Symbolic

CSE
  - Examines expressions

PRE
  - Examines expressions
  - Subsumes CSE and loop invariant code motion
  - Other implementations are now available

Constant propagation
  - Requires that values be statically known
**PRE Summary**

**What’s so great about PRE?**
- A modern optimization that subsumes earlier ideas
- Composes several simple data-flow analyses to produce a powerful result
  - Finds earliest and latest points in the CFG at which an expression is anticipated

**Next Time**

**Assignments**
- HW2 has been posted, start it now!

**Lecture**
- Alias analysis