PRE and Loop Invariant Code Motion

15-745 Spring 2007

Common Subexpression Elimination

Find computations that are always performed at least twice on an execution path and eliminate all but the first

Usually limited to algebraic expressions
  • put in some cannonical form

Almost always improves performance
  • except when?

CSE Limitation

Searches for only “totally” redundant expressions
  • An expression is totally redundant if it is recomputed along all paths leading to the redundant expression
  • An expression is partially redundant if it is recomputed along some but not all paths

Loop-Invariant Code Motion

Moves computations that produce the same value on every iteration of a loop outside of the loop

When is a statement loop invariant?
  • when all its operands are loop invariant...
Loop Invariance

An operand is loop-invariant if
1. it is a constant,
2. all defs (use ud-chain) are located outside the loop, or
3. has a single def (ud-chain again) which is inside the loop and that def is itself loop invariant

Can use iterative algorithm to compute loop invariant statements

Aside: The loop pre-header

Several optimizations require us to move statements “before the header of the loop”.

To do this we create a new block, called the pre-header, which has the loop header as successor.

All edges which formerly entered the header of L from outside of L instead enter the pre-header.

Loop Invariant Code Motion

Naïve approach: move all loop-invariant statements to the preheader

Not always valid for statements which define variables

If statement $s$ defines $v$, can only move $s$ if
- $s$ dominates all uses of $v$ in the loop
- $s$ dominates all loop exits

Loop Invariant Code Motion

Loop invariant expressions are a form of partially redundant expressions. Why?
Partial Redundancy Elimination

Moves computations that are at least partially redundant to their optimal computation points and eliminates totally redundant ones.

Encompasses CSE and loop-invariant code motion.

Optimal Computation Point

Optimal?
- Result used and never recalculated
- Expression placed late as possible

Why?

PRE Example

What expression is partially redundant?

What are the optimal computation points?

Critical Edge Splitting

In order for PRE to work well, we must split critical edges.

A critical edge is an edge that connects a block with multiple successors to a block with multiple predecessors.
Critical Edge Splitting

In order for PRE to work well, we must split critical edges

A critical edge is an edge that connects a block with multiple successors to a block with multiple predecessors.

PRE History

PRE was first formulated as a bidirectional data flow analysis by Morel and Renvoise in 1979.

Knoop, Rüthing, and Steffen came up with a way to do it using several unidirectional analysis in 1992 (called their approach lazy code motion).

- this is a much simpler method
- but it is still very complicated

PRE Example

Must compute several data flow properties in order to find optimal placement for partially redundant expressions.

Local Transparency (TRANSloc)

An expression’s value is locally transparent in a block if there are no assignments in the block to variables within the expression.

- ie, expression not killed
Local Anticipatable (ANTloc)

An expression’s value is \textit{locally anticipatable} in a block if

- there is a computation of the expression in the block
- the computation can be safely moved to the beginning of the block

\[
\begin{array}{|c|c|}
\hline
\text{Block} & \text{ANTloc} \\
\hline
\text{entry} & \emptyset \\
\text{B1} & \{ a+1 \} \\
\text{B2} & \{ x*y \} \\
\text{B2a} & \emptyset \\
\text{B3} & \emptyset \\
\text{B3a} & \emptyset \\
\text{B4} & \{ x*y \} \\
\text{B5} & \emptyset \\
\text{B6} & \emptyset \\
\text{B7} & \{ x*y \} \\
\text{exit} & \emptyset \\
\hline
\end{array}
\]

Globally Anticipatable (ANT)

An expression’s value is \textit{globally anticipatable} on entry to a block if

- every path from this point includes a computation of the expression
- it would be valid to place a computation of an expression anywhere along these paths

\textit{This is like liveness, only for expressions}

\[
\begin{array}{|c|c|c|}
\hline
\text{Block} & \text{ANTin} & \text{ANTout} \\
\hline
\text{entry} & \{ a+1 \} & \{ a+1 \} \\
\text{B1} & \{ a+1 \} & \emptyset \\
\text{B2} & \{ x*y \} & \{ x*y \} \\
\text{B2a} & \{ x*y \} & \{ x*y \} \\
\text{B3} & \emptyset & \emptyset \\
\text{B3a} & \{ x*y \} & \{ x*y \} \\
\text{B4} & \{ x*y \} & \emptyset \\
\text{B5} & \{ x*y \} & \{ x*y \} \\
\text{B6} & \emptyset & \emptyset \\
\text{B7} & \{ x*y \} & \emptyset \\
\text{exit} & \emptyset & \emptyset \\
\hline
\end{array}
\]

Globally Anticipatable (ANT)

\[\text{ANTin}(i) = \text{ANTloc}(i) \cup (\text{TRANSloc}(i) \cap \text{ANTout}(i))\]

\[\text{ANTout}(i) = \bigcap_{j \in \text{succ}(i)} \text{ANTin}(j)\]

\[\text{ANTout}(\text{exit}) = \emptyset\]

Earliest (EARL)

An expression’s value is \textit{earliest} on entry to a block if

- no path from entry to the block evaluates the expression to produce the same value as evaluating it at the block’s entry would

\textit{Intuition: at this point if we compute the expression we are computing something completely new says nothing about usefulness of computing expression}
Earliest (EARL)

\[
\text{EARLout}(i) = \overline{\text{TRANSloc}(i)} \cup (\text{ANTin}(i) \cap \text{EARLin}(i))
\]

\[
\text{EARLin}(i) = \bigcup_{j \in \text{pred}(i)} \text{EARLout}(j)
\]

\[
\text{EARLin(entry)} = \emptyset
\]

<table>
<thead>
<tr>
<th>Block</th>
<th>EARLin</th>
<th>EARLout</th>
</tr>
</thead>
<tbody>
<tr>
<td>entry</td>
<td>{a+1, x*y}</td>
<td>{x*y}</td>
</tr>
<tr>
<td>B1</td>
<td>{x*y}</td>
<td>{x*y}</td>
</tr>
<tr>
<td>B2</td>
<td>{x*y}</td>
<td>{a+1}</td>
</tr>
<tr>
<td>B2a</td>
<td>{a+1}</td>
<td>{a+1}</td>
</tr>
<tr>
<td>B3</td>
<td>{x*y}</td>
<td>{x*y}</td>
</tr>
<tr>
<td>B3a</td>
<td>{x*y}</td>
<td>{}</td>
</tr>
<tr>
<td>B4</td>
<td>{a+1}</td>
<td>{a+1}</td>
</tr>
<tr>
<td>B5</td>
<td>{a+1}</td>
<td>{a+1}</td>
</tr>
<tr>
<td>B6</td>
<td>{x*y}</td>
<td>{x*y}</td>
</tr>
<tr>
<td>B7</td>
<td>{a+1}</td>
<td>{a+1}</td>
</tr>
<tr>
<td>exit</td>
<td>{a+1, x*y}</td>
<td>{a+1, x*y}</td>
</tr>
</tbody>
</table>

Delayedness (DELAY)

An expression is \textit{delayed} on entry to a block if

- it is both anticipatable and earliest

<table>
<thead>
<tr>
<th>Block</th>
<th>DELAYin</th>
<th>DELAYout</th>
</tr>
</thead>
<tbody>
<tr>
<td>entry</td>
<td>{a+1}</td>
<td>{a+1}</td>
</tr>
<tr>
<td>B1</td>
<td>{a+1}</td>
<td>{}</td>
</tr>
<tr>
<td>B2</td>
<td>{x*y}</td>
<td>{}</td>
</tr>
<tr>
<td>B2a</td>
<td>{}</td>
<td>{}</td>
</tr>
<tr>
<td>B3</td>
<td>{}</td>
<td>{}</td>
</tr>
<tr>
<td>B3a</td>
<td>{x*y}</td>
<td>{x*y}</td>
</tr>
<tr>
<td>B4</td>
<td>{}</td>
<td>{}</td>
</tr>
<tr>
<td>B5</td>
<td>{}</td>
<td>{}</td>
</tr>
<tr>
<td>B6</td>
<td>{}</td>
<td>{}</td>
</tr>
<tr>
<td>B7</td>
<td>{}</td>
<td>{}</td>
</tr>
<tr>
<td>exit</td>
<td>{}</td>
<td>{}</td>
</tr>
</tbody>
</table>

Lateness (LATE)

An expression is \textit{latest} on entry to a block if

- it is the optimal point for computing the expression and
- on every path from the block entry to exit, any other optimal computation point occurs after an expression computation in the original flowgraph

\textit{i.e., there is no “later” placement for this expression}
Latestness (LATE)

$LATEin(i) = DELAYin(i) \cap \left(ANTloc(i) \cup \bigcap_{j \in succ(i)} DELAYin(j) \right)$

<table>
<thead>
<tr>
<th>Block</th>
<th>LATEin</th>
</tr>
</thead>
<tbody>
<tr>
<td>entry</td>
<td>{}</td>
</tr>
<tr>
<td>B1</td>
<td>{a+1}</td>
</tr>
<tr>
<td>B2</td>
<td>{x*y}</td>
</tr>
<tr>
<td>B2a</td>
<td>{}</td>
</tr>
<tr>
<td>B3</td>
<td>{}</td>
</tr>
<tr>
<td>B3a</td>
<td>{x*y}</td>
</tr>
<tr>
<td>B4</td>
<td>{}</td>
</tr>
<tr>
<td>B5</td>
<td>{}</td>
</tr>
<tr>
<td>B6</td>
<td>{}</td>
</tr>
<tr>
<td>B7</td>
<td>{}</td>
</tr>
<tr>
<td>exit</td>
<td>{}</td>
</tr>
</tbody>
</table>

Isolatedness (ISOL)

An optimal placement in a block for the computation of an expression is isolated iff

- on every path from a successor of the block to the exit block, every original computation is preceded by the optimal placement point

$ISOLin(i) = LATEin(i) \cup \left(ANTloc(i) \cap ISOLout(i) \right)$

$ISOLout(i) = \bigcap_{j \in succ(i)} ISOLin(j)$

$ISOLout(exit) = \{\}$

Optimal Placement

The set of expression for which a given block is the optimal computation point is the set of expressions that are latest and not isolated

$OPT(i) = LATEin(i) \cap \overline{ISOLout(i)}$
Redundant Computations

The set of redundant expressions in a block consist of those used in the block that are neither isolated nor latest

\[
REDN(i) = ANTloc(i) \cap LATEin(i) \cup ISOLout(i)
\]

**OPT and REDN**

<table>
<thead>
<tr>
<th>Block</th>
<th>OPT</th>
<th>REDN</th>
</tr>
</thead>
<tbody>
<tr>
<td>entry</td>
<td>{}</td>
<td>{}</td>
</tr>
<tr>
<td>B1</td>
<td>{x+y}</td>
<td>{}</td>
</tr>
<tr>
<td>B2</td>
<td>{}</td>
<td>{}</td>
</tr>
<tr>
<td>B2a</td>
<td>{}</td>
<td>{}</td>
</tr>
<tr>
<td>B3</td>
<td>{}</td>
<td>{}</td>
</tr>
<tr>
<td>B3a</td>
<td>{x+y}</td>
<td>{}</td>
</tr>
<tr>
<td>B4</td>
<td>{}</td>
<td>{x+y}</td>
</tr>
<tr>
<td>B5</td>
<td>{}</td>
<td>{}</td>
</tr>
<tr>
<td>B6</td>
<td>{}</td>
<td>{}</td>
</tr>
<tr>
<td>B7</td>
<td>{}</td>
<td>{x+y}</td>
</tr>
<tr>
<td>exit</td>
<td>{}</td>
<td>{}</td>
</tr>
</tbody>
</table>

*insert these (if necessary)*

*remove these*

**PRE Example**

4 data flow analyses later...