Reduction of operator strength

Example

```plaintext
dimension a(50,50)
...
sum = 0.0
do i = 1, 50
    sum = sum + a(3,i)
end
```

becomes

```plaintext
i ← 1
if (i > 50) goto e
l: t1 ← i × 50
t1 ← t1 + 3
t2 ← load ⟨a+t1⟩
sum ← sum + t2
i ← i + 1
if (i ≤ 50) goto l
e: ...
```

The \textit{compiler} put that multiply in the loop!

Reduction of operator strength

Definition

*Operator strength reduction* is an optimization that replaces an operator with a weaker (and, presumably, cheaper) operator.

- replacing multiply with shifts and adds (not in *your* lab)
- replacing repeated multiplies with addition

In *linpackd*, on the IBM RT/PC, strength reduction leads to an improvement of about 15 percent.

Our continuing example

```plaintext
i ← 1
t3 ← 50
if (i > 50) goto e
l:
   t1 ← t3 + 3
   t2 ← load ⟨a+t1⟩
   sum ← sum + t2
   i ← i + 1
   t3 ← t3 + 50
   if (i ≤ 50) goto l
e:
```

...
Reduction of operator strength

Linear function test replacement

Of course, there is a further simplification.

\[
\begin{align*}
t3 & \leftarrow 50 \\
\text{if } (t3 > 50) & \text{ goto } e \\
\end{align*}
\]

\[
\begin{align*}
l: & \quad t1 \leftarrow t3 + 3 \\
t2 & \leftarrow \text{load} \langle a+t1 \rangle \\
\text{sum} & \leftarrow \text{sum} + t2 \\
t3 & \leftarrow t3 + 50 \\
\text{if } (t3 \leq 2500) & \text{ goto } l \\
e: & \quad \ldots
\end{align*}
\]

Linear function test replacement shifts the test from \textit{i} to \textit{t3}.

- often eliminates last use for iteration variable
- fewer instructions, fewer live ranges
Reduction of operator strength

Assumptions

- intermediate code is quadruples
- have built the control flow graph
- have found natural loops or strongly connected regions
- have added a “prolog” or “landing pad” to each region

Definitions

- a region constant is a variable whose value is unchanged throughout the SCR (global constants)
- an induction variable is a variable whose value is changed in the SCR only by instructions that increment it by a region constant
Reduction of operator strength

The problem

• easy to “see” what to do
• hard to automate the process

The big picture \((within a \, SCR)\)

1. find all the induction variables
2. find instructions that can be reduced \((candidates)\)
3. find all the quantities that affect their values
4. \((finally,)\) perform the actual replacement
Reduction of operator strength

Step 1: Finding induction variables

**Input:** SCR - set of instructions in the region
   RC, the set of region constants

**Output:** IV, the set of induction variables

IV ← ∅
for each instruction i in SCR do
   (t ← a1 op a2)
   if op ∈ {add, sub, neg, load, store}  
      IV ← IV ∪ t

changed ← true
while (changed)
   changed ← false
   for each instruction i where t ∈ IV
      if a1 ∉ (IV ∪ RC) or a2 ∉ (IV ∪ RC)
         remove t from IV
         changed ← true
      if op ∉ {add, sub, neg, load, store}
         remove t from IV
         changed ← true
Reduction of operator strength

Step 2: Finding candidate instructions

*Input:* SCR – set of instructions in the region
IV and RC

*Output:* CANDS - the set of candidate instructions

\[ \text{CANDS} \leftarrow \emptyset \]

\[ \text{for each instruction } i \quad (t \leftarrow a_1 \ op \ a_2) \]

if \( op \) is multiply \&
\[ (a_1 \in \text{IV} \ & \ a_2 \in \text{RC} \ \text{or} \]
\[ a_1 \in \text{RC} \ & \ a_2 \in \text{IV}) \]
\[ \text{CANDS} \leftarrow \text{CANDS} \cup \{i\} \]

CANDS contains all multiplies that involve exactly one region constant and one induction variable.
Reduction of operator strength

Naming

- create a temporary for each unique candidate expression
  — hash them for uniqueness
- insert an initialization for each temporary
  — place it in the landing pad
- after an assignment to $i \in IV$, insert an update to the appropriate temporaries

<table>
<thead>
<tr>
<th>Reducing $x \leftarrow i \times c$</th>
<th>Operation to be Inserted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction</td>
<td></td>
</tr>
<tr>
<td>$i \leftarrow k$</td>
<td>$t_{ixc} \leftarrow t_{kxc}$</td>
</tr>
<tr>
<td>$i \leftarrow -k$</td>
<td>$t_{ixc} \leftarrow -t_{kxc}$</td>
</tr>
<tr>
<td>$i \leftarrow j + k$</td>
<td>$t_{ixc} \leftarrow t_{jxc} + t_{kxc}$</td>
</tr>
<tr>
<td>$i \leftarrow j - k$</td>
<td>$t_{ixc} \leftarrow t_{jxc} - t_{kxc}$</td>
</tr>
</tbody>
</table>

To deal with all of these cases, we associate with each $i \in IV$ a set $AFFECT(i)$ that contains every $j \in IV \cup RC$ that can affect the value of $i$. 
Reduction of operator strength

Step 3: Computing AFFECT sets

Input: SCR – set of instructions in the region
       IV – set of induction variables

Output: AFFECT(i) for each \( i \in IV \)

for each \( i \in IV \)
   \( \text{AFFECT}(i) \leftarrow i \)

for each instruction \( j \) where \( t \in IV \quad (t \leftarrow a_1 \text{ op } a_2) \)
   \( \text{AFFECT}(t) \leftarrow \text{AFFECT}(t) \cup \{a_1, a_2\} \)

\( \text{changed} \leftarrow \text{true} \)

while (\( \text{changed} \))
   \( \text{changed} \leftarrow \text{false} \)
   for each \( i \in IV \)
      \( \text{NEW} \leftarrow \bigcup_{j \in \text{AFFECT}(i) \cap IV} \text{AFFECT}(j) \)
      if \( \text{AFFECT}(i) \cap \text{NEW} \neq \emptyset \)
         \( \text{changed} \leftarrow \text{true} \)
         \( \text{AFFECT}(i) \leftarrow \text{AFFECT}(i) \cup \text{NEW} \)
Reduction of operator strength

Step 4: Replacement

/* build up a set of multipliers for variable */
for each $x \in \text{IV} \cup \text{RC}$
    \[
    \text{CLIST}(x) \leftarrow \emptyset
    \]

for each instruction $p \in \text{CANDS}$
    \[
    (x \leftarrow i \times c, \text{ with } x \in \text{IV} \text{ and } c \in \text{RC})
    \]
    for each $y \in \text{AFFECT}(x)$
        \[
        \text{CLIST}(y) \leftarrow \text{CLIST}(y) \cup c
        \]

/* initialize each reduced induction variable */
for each $x \in \text{IV} \cup \text{RC}$ with $\text{CLIST}(x) \neq \emptyset$
    for each $c \in \text{CLIST}(x)$
        \[
        T(x, c) \leftarrow \text{new temporary name}
        \]

        insert “$T(x, c) \leftarrow \text{mult } x \ c$” at the landing pad

/* insert updates for each reduced induction variable */
for each instruction $p$ with $t \in \text{IV}$ and $\text{CLIST}(t) \neq \emptyset$
    for each $c \in \text{CLIST}(t)$
        insert after $p$ (op is same as in $p$)
            “$T(t, c) \leftarrow T(a_1, c) \ op \ T(a_2, c)$”

/* replace the candidate instructions */
for each instruction $p \in \text{CANDS}$
    (assume $x \in \text{IV}$ and $c \in \text{RC}$)
    replace $p$ with the instruction “$t \leftarrow T(x, c)$”
Linear function test replacement

**Input:** SCR and its landing pad
IV, RC, and CLIST
hash table from replacement algorithm

**Output:** modified SCR and its landing pad

∀ instruction in SCR
if it is a conditional branch (bc label i op k)
with i ∈ IV and k ∈ RC
select some c ∈ CLIST(i)
if neither T(c,k) nor T(k,c) exist
assign T(c,k) a name
insert “T(c,k) ← c × k” in landing pad

Replace the conditional branch with
bc label T(i,c) op T(k,c)

**Notes:**

1. if CLIST(i) = ∅, we cannot replace the test.
2. we can choose any c ∈ CLIST(i)
Reduction of operator strength

Room for improvement

- reduce other operators
- enlarge set of region constants
- speed up the algorithm
- integrate with expression reassociation

And, of course, . . .

- strong dead code elimination, or
- inserting less dead code