Strength Reduction
Strength reduction

• Like strength reduction peephole optimization
• Peephole: replace expensive instruction like \(a \times 2\) with \(a \ll 1\)
• Replace expensive instruction, multiply, with a cheap one, addition
• Applies to uses of an \textit{induction variable}
• Opportunity: array indexing

```c
for (i = 0; i < 100; i++)
    A[i] = 0;
```

for (i = 0; i >= 100) goto L1
    
    j = 4 * i + &A
    *j = 0;
    i = i + 1;
    goto L2
```

L1:
Strength reduction

• Like strength reduction peephole optimization
  • Peephole: replace expensive instruction like \( a \times 2 \)
    with \( a \ll 1 \)
  • Replace expensive instruction, multiply, with a cheap one, addition
  • Applies to uses of an \textit{induction variable}
• Opportunity: array indexing

\[
\begin{align*}
\text{for} \ (i = 0; i < 100; i++) & \quad A[i] = 0; \\
& \quad i = 0; \ k = \&A; \\
& \text{L2: if} \ (i \geq 100) \ \text{goto L1} \\
& \quad j = k; \\
& \quad *j = 0; \\
& \quad i = i + 1; \ k = k + 4; \\
& \ \text{goto L2} \\
& \text{L1:}
\end{align*}
\]
Induction variables

- A *basic induction variable* is a variable $i$ whose only definition within the loop is an assignment of the form $i = i \pm c$, where $c$ is loop invariant.

- Intuition: the variable which determines the number of iterations is usually an induction variable.

- A *mutual induction variable* $j$ may be:
  - defined once within the loop, and its value is a linear function of some other induction variable $i$ such that $j = c_1 \times i \pm c_2$ or $j = i/c_1 \pm c_2$
  - where $c_1$, $c_2$ are loop invariant.

- A *family* of induction variables include a basic induction variable and any related mutual induction variables.
Strength reduction algorithm

• Let j be an induction variable in the family of the basic induction variable i, such that \( j = c_1 \times i + c_2 \)

• Create a new variable j’

• Initialize in preheader

\[
j' = c_1 \times i + c_2
\]

• Track value of i. After \( i = i + c_3 \), perform

\[
j' = j' + (c_1 \times c_3)
\]

• Replace definition of j with

\[
j = j'
\]

• Key: c1, c2, c3 are all loop invariant (or constant), so computations like \( c_1 \times c_3 \) can be moved outside loop
Linear test replacement

- After strength reduction, the loop test may be the only use of the basic induction variable.
- Can now eliminate induction variable altogether.
- Algorithm
  - If only use of an induction variable is the loop test and its increment, and if the test is always computed,
  - Can replace the test with an equivalent one using one of the mutual induction variables.

```
i = 2
for (; i < k; i++)
j = 50*i
... = j
```

Strength reduction
```
i = 2; j' = 50 * i
for (; i < k; i++, j' += 50)
... = j'
```

Linear test replacement
```
i = 2; j' = 50 * i
for (; j' < 50*k; j' += 50)
... = j'
```
Loop unrolling

• Modifying induction variable in each iteration can be expensive

• Can instead unroll loops and perform multiple iterations for each increment of the induction variable

• What are the advantages and disadvantages?
  
  • fewer instructions executed, more opportunities for CSE, strength reduction, ILP etc.
  
  • code size increase, more i-cache pressure, can confuse allocator

for (i = 0; i < N; i++)
A[i] = ...

Unroll by factor of 4

for (i = 0; i < N; i += 4)
A[i] = ...
A[i+1] = ...
A[i+2] = ...
A[i+3] = ...
next: high-level loop optimization