Induction Variables

Announcements
- HW1 due Friday

Last Time
- Code Motion

Today
- Induction variables

Induction Variables

Induction variable identification
- Induction variables
  - Variables whose values form an arithmetic progression

Why bother?
- Useful for strength reduction, induction variable elimination, loop transformations, and automatic parallelization

Simple approach
- Search for statements of the form, $i = i + c$
- Examine ud-chains to make sure there are no other defs of $i$ in the loop
- Does not catch all induction variables. Examples?

Example Induction Variables

```c
s = 0;
for (i=0; i<N; i++)
  s += a[i];
```
Induction Variable Triples

Each induction variable \( k \) is associated with a triple \((i, c_1, c_2)\)
- \( i \) is a basic induction variable
- \( c_1 \) and \( c_2 \) are constants such that \( k = c_1 + c_2 \cdot i \) when \( k \) is defined
- \( k \) belongs to the family of \( i \)

Basic induction variables
- their triple is \((i, 0, 1)\)
- \( i = 0 + 1 \cdot i \) when \( i \) is defined

Algorithm for Identifying Induction Variables

Input: A loop \( L \) consisting of 3-address instructions, ud-chains, and loop-invariant information.
Output: A set of induction variables, each with an associated triple.
Algorithm:
1. For each stmt in \( L \) that matches the pattern \( i = i + c \) or \( i = i - c \)
   create the triple \((i, 0, 1)\).
2. Derived induction variables: For each stmt of \( L \),
   - If the stmt is of the form \( k = j + c_1 \) or \( k = j \cdot c_2 \)
     and \( j \) is an induction variable with the triple \((x, p, q)\)
     and \( c_1 \) and \( c_2 \) are loop invariant
     and \( k \) is only defined once in the loop
     and if \( j \) is a derived induction variable belonging to the family of \( i \) then
       the only def of \( j \) that reaches \( k \) must be in \( L \)
       and no def of \( i \) must occur on any path between the definition of \( j \) and \( k \)
     then create the triple \((x, p + c_1, q)\) for \( k = j + c_1 \)
     or \((x, p \cdot c_2, q \cdot c_2)\) for \( k = j \cdot c_2 \)

Example: Induction Variable Detection

![Example Diagram]

Picture from Prof David Walker’s CS320 slides

Algorithm for Strength Reduction

Input: A loop \( L \) consisting of 3-address instructions and induction variable triples.
Output: A modified loop with a new preheader.
Algorithm:
1. For each derived induction variable \( j \) with triple \((i, p, q)\)
   create a new \( j' \)
   after each definition of \( i \) in \( L \), where \( i = i + c \) put computation \( sq \cdot t \) in preheader
   replace the definition of \( j \) with \( \text{j' = j' + i} \)
   initialize \( j' \) at the end of the preheader to \( j' = p + q \cdot t \)

Note:
- \( j' \) also has triple \((i, p, q)\)
- multiplication has been moved out of the loop
**Algorithm for Induction Variable Elimination**

**Input:** A loop L consisting of 3-address instructions, ud-chains, loop-invariant information, and live-variable information.

**Output:** A revised loop.

**Algorithm:**

1. For each induction variable $i$
   - If only use is define itself and is dead out of loop exit nodes, then mark as eliminated
   - else if only uses are to compute other induction variables in its family and in conditional branches, then mark as eliminated
     - Use a triple $(j, c, d)$ in family associated with variable $k$
     - Modify each conditional involving $i$ so that $k$ is used instead, uses relationships set up with triples
     - Delete all assignments to the eliminated induction variable
2. Apply copy propagation followed by dead code elimination to eliminate copies introduced by strength-reduction.
3. Remove any induction variable definitions where the induction variable is only used and defined within that definition.

---

**Example: Strength Reduction**

```
1:  i1 = 0
2:  i2 = 0
3:  branch (l2 = N)
4:  i3 = i2 * 1
5:  i4 = i3 + a
6:  i5 = i4 + c
7:  l1 = i1 + e
8:  i2 = i2 + 1
9:  jump
```

---

**Summary**

**Induction variable detection uses**
- strength reduction and induction variable elimination
- data dependence analysis, which can then be used for parallelization

**Strength reduction**
- removes multiplications
- the definition for some derived induction variables no longer depend directly on a basic induction variable

**Induction variable elimination**
- removes unnecessary induction variables

**Next Time**

**Lecture**
- Midterm Review