Induction Variables & Strength Reduction

Induction variable: variable whose successive values form an arithmetic progression in loop

Ex. do i = 1,99
    a(i) := 2*i - 1
enddo

i is inductive
2*i-1 is inductive

t := -1
do i = 1,99
    t := t + 2
    a(i) := t
enddo

Replaces expensive ops (like *) with less expensive ops (like +)

Strength Reduction

Loop Peeling

Remove 1 or more iterations from beginning/end of loop

Ex. do i = 1,100
    a(i) := a(i) + a(1)
enddo

i = 1
i > 1

a(1) := a(1) + a(1)
doi = 2,100
a(i) := a(i) + a(1)
enddo

Beneficial?

Always possible?
**Index Set Splitting**

Split index set 1,...,N into multiple sets

Ex. do i = 1,100
    a(i) := b(i) + c(i)
    if (i>50) then
        d(i) := a(i) + a(i-10)
    enddo

Beneficial?

Always possible?
Global Value Numbering with SSA

X, Y are \textit{dynamically equivalent} at P if they have the same values whenever control reaches P on execution.

Undecidable \xrightarrow{} Develop static notion \textit{Congruence}

X congruent to Y \xrightarrow{} X dynamically equivalent to Y

Go beyond Basic Block

Ex.

Value Graph for Basic Block

A := 3
B := 3
C := A+1
D := B+1
if (C>3)...

\texttt{C and D are congruent:}

\textit{have identical operators, and like operands are congruent}

Like value-numbering
Why SSA?

J := 5
K := 5

J := 6
K := 7

J1 := 5
K1 := 5

J2 := 6
K2 := 7

congruent

not congruent

J1, K1 congruent
J2, K2 not congruent

What about control flow?

I1 := 5

I1, J1, K1 congruent at B if assignments dominate B
Value Graph for SSA

Nodes  Constants, operators, phi-functions

Directed Edges  From use to node where value generated

Labels  Constant, operator, function symbols

Ex.  

\[(I_1 > 29)\]
\[J_1 := 1\]
\[K_1:= 1\]
\[J_3 := \phi (J_1, J_2)\]
\[K_3 := \phi (K_1, K_2)\]

\[\text{BB} \# \text{for phi function}\]

---

Value Graph for SSA (example)
**Congruence**

A is **congruent** to node B if

1. A is the same node as B, or
2. A and B are constant nodes, with the same constant value, or
3. A and B are operator nodes, with the same operator, and their like operands are congruent

Vars X and Y are **equivalent** at P if their nodes are congruent and defining assignments dominate P.

**Ex.**

\[ J_1, K_1, L_1 \text{ congruent} \]
\[ J_2, K_2, L_2 \text{ congruent} \]
\[ J_3, K_3 \text{ congruent, but not with } L_3 \]

Get equivalence classes of variables

---

**Loop Example**

Entry
\[ \text{read}(N_1) \]
\[ J_1 := 1 \]
\[ I_1 := 1 \]

\[ J_3 := \phi(J_1,J_2) \] (2)
\[ I_3 := \phi(I_1,I_2) \]
\[ (I_3 \mod 2 = 0) \]

\[ J_4 := J_3 + 1 \]
\[ I_4 := I_3 + 1 \] (3)
\[ J_5 := J_3 + 2 \]
\[ I_5 := I_3 + 2 \]

\[ J_2 := \phi(J_4,J_5) \]
\[ I_2 := \phi(I_4, I_5) \]
\[ (J_2 > N_1) \] Exit

Value graph for J2 is identical

**BUT cycles prohibit finding it!**
**Algorithm Overview**

1. Compute SSA.
2. Build value graph for SSA.
3. Optimistically assume all nodes with same label are congruent. Determine congruence of nodes by partitioning algorithm.
4. Check for equivalence.

**Partitioning:** \( (O(E \log E)) \)

1. Put all nodes with same label in same partition.
2. **i+1:** Two nodes are in same partition at step \( i+1 \), if at step \( i \), they are in the same partition and the destination of their edges are in the same partition.

**Taking Control Flow into account**

![Diagram](image-url)
Loop Example

Detecting Congruence:
1. Same initial values
2. Same modifications in loop
3. Same no. of iterations

Entry
→
read(N1)
J1 := 1
I1 := 1

J3 := φ(J1, J2) (J1, J2)
I3 := φ(I1, I2)
(I3 mod 2 = 0)

J4 := J3 + 1
I4 := I3 + 1

J5 := J3 + 2
I5 := I3 + 2

J2 := φ(J4, J5)
I2 := φ(I4, I5)
(J2 > N1)

J6 := φ(φ(φ(J2 > N1, J2)))

Other Extensions

Incorporate arrays, pointers

*Update, Access functions*

Take commutativity into account

*Ex. a*b same as b*a*

Combine with hash-based approach

*(Cooper et. al.)*