Module 16: Data Flow Analysis in Presence of Procedure Calls

Lecture 31: Data Dependence Analysis

The Lecture Contains:

- Interprocedural Dataflow Analysis
- Alias Computation
- Example
- Data Flow Analysis in Presence of Procedure Calls
- Data Dependence Analysis
- Data Dependence
- Data Dependence Graph
- Basic Block Dependence
- Data Dependence in Loops
- Unroll the Loop

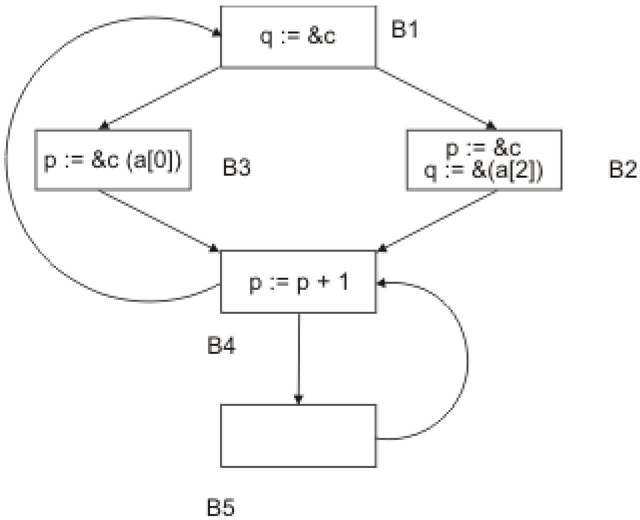
Previous Next

We can relate in, out and transfer as follows

$$in[\,B\,] \quad \bigcup_{\text{P is pred of B}} \quad out[\,P\,]$$

$$out[\,B\,] \quad trans_B\,(in[\,B\,])$$

Consider following control flow graph



Suppose a is an array, c is and integer, and p and q are pointer.

Initially $[B_1] = \emptyset$

Out $[\mathrm{B}_1] = \mathrm{trans}_{\mathrm{B}_2}(\{(\mathrm{q},\mathrm{c})\}) = \{(\mathrm{p}.\,\mathrm{c}),(\mathrm{q},\mathrm{a})\}$

In $[B_2] = out\ [B_1]$

out $[\mathrm{B}_2] = \mathrm{trans}_{\mathrm{B}_2}(\{(\mathrm{q},\mathrm{c})\}) = \{(\mathrm{p},\mathrm{c}),(\mathrm{q},\mathrm{a})\}$

In $[B_3] = out\ [B_1]$

out $[\mathrm{B}_3] = \mathrm{trans}_{\mathrm{B}_3}(\{(\mathrm{q},\mathrm{c})\}) = \{(\mathrm{p},\mathrm{c}),(\mathrm{q},\mathrm{a})\}$

in $[B_4] = out[B_2] \cup out\ [B_3] \cup out\ [B_5]$

in $[\mathrm{B}_4] = \{(\mathrm{p},\mathrm{a}),(\mathrm{p},\mathrm{c}),(\mathrm{q},\mathrm{a}),(\mathrm{q}.\,\mathrm{c})\}$

out $[B_4] = trans_{B4}(in[B_4]$

=\{(P,a),(q,a),(q,c)

in $[\mathrm{B}_5] = out\ [B_4]$

out $[B_5] = \{(\mathrm{p},\mathrm{a}),(\mathrm{p},\mathrm{c}),(\mathrm{q},\mathrm{a}),(\mathrm{q}.\,\mathrm{c})\}$

Previous Next

Interprocedural dataflow analysis

Aliases : If two variables denote the same memory location

s1 : a := b+x
s2 : y := c
s3 : d := b+x

is b+x available at s3?
Yes, provided x and y are not aliases

language :

- Permits recursive procedures
- May refer to both global & local definitions
- Data variables consist of globals and its own locals (no block structuring)
- Parameters by reference
- Single return node

Alias Computation

1. Rename variables so that no two procedures use the same formal parameters or local identifiers
2. If there is a procedure $P(X_1 \ldots X_n)$ and an invocation $P(Y_1 \ldots Y_n)$, set $X_i \equiv Y_i$
3. Take reflexive and transitive closure by adding
   X = Y whenever Y = X
   X = Z whenever X = Y and Y = Z

Example

global g,h
zero();
local i;
g := · · ·
one(h, i); h = w i = x
end zero;
one(w, x)
x := · · ·
two(w, w); w = y w = z
two(g, x); g = y x = z
end one;
two(y, z)
local k;
h := · · ·
one(k, y) k = w y = x
end two;
Therefore, h = w = y = z = k = x = i = g

Previous Next

Data Flow Analysis in Presence of Procedure Calls

Change[p] set of global variables and formal parameters of p that might be changed during an execution of p.
Def[p] set of formal parameters and global variables having explicit definition within p.
A: { a | a is a global variable or formal of p, such that for some procedure q and integer i, p calls q with a as the ith actual parameter and ith formal of q is in change[q] }
G: { g | g is a global in change[q] and p calls q }

$$\text{change }[p] = \text{def}[p] \cup A \cup$$

Data Dependence Analysis

- Used for instruction scheduling
- Used for data cache optimization
- Determines ordering relationship; a dependence between two statements constraints their execution order
- Control dependence: arises from control flow

S1: a = b+c
S2: if a > 10 goto L1
S3: d = b*e
S4: e = d+1
S5: L1: d = e/2

Data Dependence

- Arises from *flow* of data between two statements
- Compiler must analyze programs to find *constraints* preventing the reordering of operations.

Consider:

A = 0	(1)
B = A	(2)
C = A + D	(3)
D = 2	(4)

- Moving (2) above (1):: Value of A in (2) changes
- Moving (4) above (3):: results in wrong value of D in (3)

Previous Next

Three types of constraints:

- **Flow or True Dependence**: When a variable is assigned or defined in one statement and used in subsequent statement
- **Anti Dependence**: When a variable is used in one statement and reassigned in subsequently executed statement
- **Output Dependence**: When a variable is assigned in one statement and reassigned in subsequent statement

Anti dependence and *Output* dependence arise from reuse of variable and are also called False dependence.

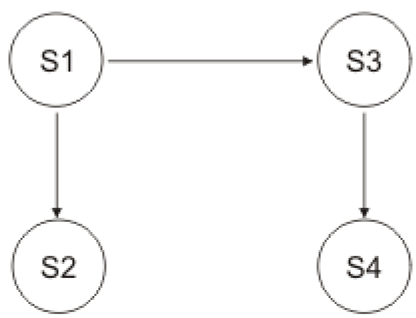
Flow dependence is inherent in computation and cannot be eliminated by renaming. Therefore it is also called True dependence.

Data Dependence Graph

Data structure used to depict dependency between statements.

- Each statement represents a node in the graph
- Nodes are connected by directed edges

1. When S2 is flow dependent on S1, it is denoted by S1 δ^f S2 or S1 δ S2 and represented by
   S1 → S2
2. When there is an anti-dependence from S1 to S2, it is denoted by S1 $\bar{\delta}$ S2 or S1 δ^a S2 and represented by
   S1 → S2
3. When there is an output-dependence from S1 to S2, it is denoted by S1 δ^0 S2 and represented by
   S1 → S2



Previous Next

Approaches to data dependence relations:

- Address Based: Dependences which use the same address
- Value Based: Dependences which use the same value

Consider A = 0
B = A
A = B + 1
C = A

Address-based approach	*Value-Based Approach*
There is a flow dependence between S1 and S4 S1 → S4 because S4 uses A	In (4) value of A used is defined in (3) and not in (1) thus, there is no data dependence

Value based dependence is a subset of Address based dependence.

For Address-based dependence:

$$out\,(S_1) \cap in\,(S_2)\ \neq\ \emptyset\ \Longrightarrow\ S_1\,\delta^f\,s_2$$

$$in\,(S_1) \cap out\,(S_2)\ \neq\ \emptyset\ \Longrightarrow\ S_1\,\delta^a\,s_2$$

$$out\,(S_1) \cap out\,(S_2)\ \neq\ \emptyset\ \Longrightarrow\ S_1\,\delta^\circ\,s_2$$

$if\ in\,(s_1) \cap in(s_2) \neq \emptyset$ then ?

This is written as $s_1\ \delta^i\,s_2$ and is used for cache *optimizations*

Basic Block Dependence

- Construct dependence graph for the instructions
- I_1 and I_2 may have flow, anti or output dependence
- Can not determine whether I_1 can be moved beyond I_2
- Suppose an instruction reads from $[r_{11}](4)$ and the next instruction writes to $[r_{12}+12](4)$
- Unless we know r_{11} and $r_{12}+12$ point to different locations assume a flow dependence
- I_1 is a predecessor of I_2 if I_2 must not execute before some cycles of I_1
- Type of dependency is not important

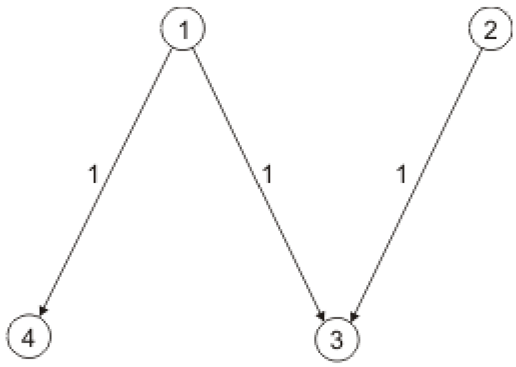
Module 16: Data Flow Analysis in Presence of Procedure Calls

Lecture 31: Data Dependence Analysis

Latency: delay required between initiation times of I_1 and I_2 minus execution time required for I_1 before another instruction can start. For example, if two cycles must elapse between I_1 and I_2 then latency is 1.

r2 ← [r1](4)
r3 ← [r1+4](4)
r4 ← r2 + r3
r5 ← r2 - 1

assume load has latency of 1; requires 2 cycles to finish.



Data Dependence in Loops

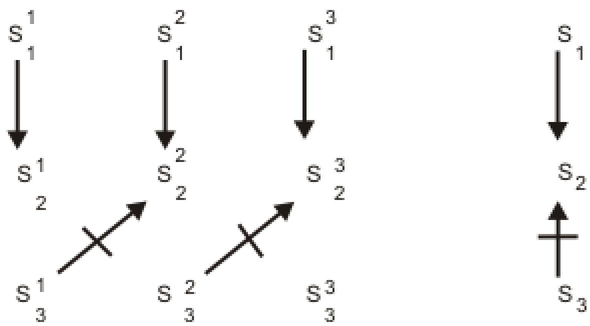
- Each statement executed many times
- Dependence can flow from one statement to any other
- Dependence can flow to the same statement

for i = 2, 9 do
x(i) = y(i) + z(i) S_1
a(i) = x(i-1) + 1 S_2
endfor
$S_1 \rightarrow S_2$

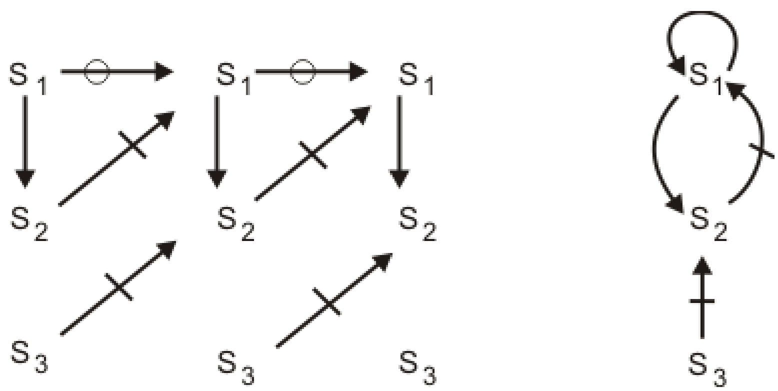
Previous Next

```
do i = 1, N
a(i) = b(i)
c(i) = a(i) + b(i)
e(i) = c(i+1)
enddo
```

Unroll The Loop



```
do I = 1, N
A = B(I)
C(I) = A + B(I)
E(I) = C(I+1)
enddo
```



Module 16: Data Flow Analysis in Presence of Procedure Calls

Lecture 31: Data Dependence Analysis

A data dependence is loop independent if dependence is between instances in the same iteration
Consider a loop

do i = 1, N
X(f(i)) = ... S_1
...... = X(g(i)) S_2
enddo

there is a loop independent dependence from S_1 to S_2 if there is an integer i such that

1 = i = N and f(i) = g(i)

OR

there is an iteration in which S_1 writes into X and S_2 reads from the same element of X.

A data dependence is loop dependent if dependence is between different iterations.

there is a loop dependent dependence from S_1 to S_2 if there exist integers i_1 and i_2 such that

$1 = i_1 < i_2 = N$ and $f(i_1) = g(i_2)$

OR

S_1 writes into X in iteration i_1 and S_2 reads from the same location in a later iteration i_2.

Therefore, to find out data dependence from S_1 to S_2, one has to solve

$f(i_1) = g(i_2)$
such that $1 = i_1 = i_2 = N$ holds.

Previous Next

Example:

do I = 1, 100
X(2I+1) = ... S_1
...... = X(2I+4) S_2
enddo

- Is there a dependence from S_1 to S_2 ?
- Coarse grain analysis: S_1 writes into X and S_2 reads from X. Therefore,
  $S_1 \rightarrow S_2$ or $S_1\ \delta^f\ S_2$
- Fine grain analysis:
  Eqn: $2i_1 + 1 = 2i_2 + 4$
  has no integer solution
  Therefore, no dependence from S_1 to S_2

DO I = 1, 50
X(I) = ...
.... = X(I+50)
ENDDO

- Fine grain analysis:
  Eqn. $i_1 = i_2 + 50$ has integer solution.
- However, no integer solution in the range
  $1 = i_1 = i_2 = 50$
  therefore, no dependence from S_1 to S_2
- If f and g are general functions, then the problem is intractable.
- If f and g are linear functions of loop index, then to test dependence we need to find values of two integers i_1 and i_2 such that

$1 \leq i_1 \leq i_2 \leq N$
and $a_0 + a_1 i_1 = b_0 + b_1 i_2$
which can be rewritten as
$1 \leq i_1 \leq i_2 \leq N$
$a_1 i_1 - b_1 i_2 = b_0 - a_0$
These are called Linear *Diophantine Equations.*

Previous Next