Handling Control Flow

Aniket Dalal-05305403
Kumar N-05305027
Vishal Khandelwal-05305404

7^{th} November, 2005

What we have seen so far..

- Data dependences
- Control dependences (keep what we have learnt about this aside)

What we have seen so far..

- Data dependences
- Control dependences (keep what we have learnt about this aside)

Why control dependence ?

$$
\begin{array}{ll}
 & DO\ I = 1, 10 \\
S1: & \quad If(A[I-1] > 0)\ \ GOTO\ 100 \\
S2: & \quad A[I] = A[I] + 5 \\
100 & ENDO
\end{array}
$$

$$\Downarrow$$

$$
\begin{array}{ll}
S2: & A[1:10] = A[1:10] + 5 \\
 & DO\ I = 1, 10 \\
S1: & \quad If(A[I-1] > 0)\ \ GOTO\ \ 100 \\
100 & ENDO
\end{array}
$$

Why control dependence ?

$$
\begin{array}{ll}
 & DO\ I = 1, 10 \\
S1: & \quad If(A[I-1] > 0)\ \ GOTO\ 100 \\
S2: & \quad A[I] = A[I] + 5 \\
100 & ENDO
\end{array}
$$

$$\Downarrow$$

$$
\begin{array}{ll}
S2: & A[1:10] = A[1:10] + 5 \\
 & DO\ I = 1, 10 \\
S1: & \quad If(A[I-1] > 0)\ \ GOTO\ \ 100 \\
100 & ENDO
\end{array}
$$

Options

- Handling control flow
  - Convert control dependences to data dependences
  - Handle Control dependences separately (what has already been covered)
- We focus on **Converting control dependences to data dependences**.

Considering the previous example again:

$$
\begin{array}{ll}
 & DO\ I = 1, 10 \\
S1: & \quad If(A[I-1] > 0)\ \ GOTO\ 100 \\
S2: & \quad A[I] = A[I] + 5 \\
100 & ENDO
\end{array}
$$

$\Downarrow$

$$
\begin{array}{ll}
 & DO\ I = 1, 10 \\
S2: & \quad If(A[I-1] \leq 0)\ \ A[I] = A[I] + 5 \\
 & ENDO
\end{array}
$$

Considering the previous example again:

$$
\begin{array}{ll}
 & DO\ I = 1, 10 \\
S1: & \quad If(A[I-1] > 0)\ \ GOTO\ 100 \\
S2: & \quad A[I] = A[I] + 5 \\
100 & ENDO
\end{array}
$$

$$\Downarrow$$

$$
\begin{array}{ll}
 & DO\ I = 1, 10 \\
S2: & \quad If(A[I-1] \leq 0)\ \ A[I] = A[I] + 5 \\
 & ENDO
\end{array}
$$

Definition

If-Conversion is the process of removing all *branches* from a program. By *branches*, we mean:

- Forward branch
- Backward branch
- Exit branch

If-Conversion is a composition of two different transformations:

- *Branch relocation*: move branches out of loop
- *Branch removal*: eliminate forward branches

Definition

If-Conversion is the process of removing all *branches* from a program. By *branches*, we mean:

- Forward branch
- Backward branch
- Exit branch

If-Conversion is a composition of two different transformations:

- *Branch relocation*: move branches out of loop
- *Branch removal*: eliminate forward branches

Forward branch

Transfers control to a target that occurs after the branch but at the same loop nesting level. E.g.

```
100   DO I = 1, 10
C1 :      If(A[I] > 10)  GOTO S3
S1 :      A[I] = A[I] + 10
C2 :      If(B[I] > 10)  GOTO S4
S2 :      B[I] = B[I] + 10
S3 :      A[I] = A[I] + B[I]
S4 :      B[I] = A[I] − 5
      ENDO
```

⇓

```
100   DO I = 1, 10
          m1 = A[I] > 10
S1 :      If(!m1)  A[I] = A[I] + 10
          If(!m1)  m2 = B[I] > 10
S2 :      If((!m1) AND (!m2))  B[I] = B[I] + 10
S3 :      If(((!m1) AND (!m2)) OR (m1))  A[I] = A[I] + B[I]
S4 :      If(((!m1) AND (!m2)) OR (m1) OR ((!m1) AND (m2)))  B[I] = A[I] − 5
      ENDO
```

Forward branch

Transfers control to a target that occurs after the branch but at the same loop nesting level. E.g.

$$
\begin{array}{ll}
100 & DO\ I = 1, 10 \\
C1: & \quad If(A[I] > 10)\ \ GOTO\ S3 \\
S1: & \quad A[I] = A[I] + 10 \\
C2: & \quad If(B[I] > 10)\ \ GOTO\ S4 \\
S2: & \quad B[I] = B[I] + 10 \\
S3: & \quad A[I] = A[I] + B[I] \\
S4: & \quad B[I] = A[I] - 5 \\
 & ENDO
\end{array}
$$

$$\Downarrow$$

$$
\begin{array}{ll}
100 & DO\ I = 1, 10 \\
 & \quad m1 = A[I] > 10 \\
S1: & \quad If(!m1)\ \ A[I] = A[I] + 10 \\
 & \quad If(!m1)\ \ m2 = B[I] > 10 \\
S2: & \quad If((!m1)\ AND\ (!m2))\ \ B[I] = B[I] + 10 \\
S3: & \quad If(((!m1)\ AND\ (!m2))\ OR\ (m1))\ \ A[I] = A[I] + B[I] \\
S4: & \quad If(((!m1)\ AND\ (!m2))\ OR\ (m1)\ OR\ ((!m1)\ AND\ (m2)))\ \ B[I] = A[I] - 5 \\
 & ENDO
\end{array}
$$

Forward branch

Transfers control to a target that occurs after the branch but at the same loop nesting level. E.g.

```
100   DO I = 1, 10
C1 :      If(A[I] > 10)  GOTO S3
S1 :      A[I] = A[I] + 10
C2 :      If(B[I] > 10)  GOTO S4
S2 :      B[I] = B[I] + 10
S3 :      A[I] = A[I] + B[I]
S4 :      B[I] = A[I] − 5
      ENDO
```

⇓

```
100   DO I = 1, 10
          m1 = A[I] > 10
S1 :      If(!m1)  A[I] = A[I] + 10
          If(!m1)  m2 = B[I] > 10
S2 :      If((!m1) AND (!m2))  B[I] = B[I] + 10
S3 :      If(((!m1) AND (!m2)) OR (m1))  A[I] = A[I] + B[I]
S4 :      If(((!m1) AND (!m2)) OR (m1) OR ((!m1) AND (m2)))  B[I] = A[I] − 5
      ENDO
```

Exit branch

Terminates one or more loops by transferring control to a target outside a loop. E.g.

```
      DO J = 1, M
        DO I = 1, N
          A[I, J] = B[I, J] + 10
S :       IF(L[I, J]) GOTO 200
          C[I, J] = A[I, J] + 10
        ENDO
        D[J] = A[N, J]
200 :   F[J] = C[10, J]
      ENDO
```

$\Rightarrow$

```
      DO J = 1, M
        lm = TRUE
        DO I = 1, N
          IF(lm) A[I, J] = B[I, J] + 10
          IF(lm) m1 = !L[I, J]
          lm = lm AND m1
          IF(lm) C[I, J] = A[I, J] + 10
        ENDO
        m2 = lm
        IF(m2) D[J] = A[N, J]
200 :   F[J] = C[10, J]
      ENDO
```

Exit branch

Terminates one or more loops by transferring control to a target outside a loop. E.g.

$$
\begin{array}{ll}
 & DO\ J = 1, M \\
 & \quad DO\ I = 1, N \\
 & \quad\quad A[I,J] = B[I,J] + 10 \\
S: & \quad\quad IF(L[I,J])\ \ GOTO\ 200 \\
 & \quad\quad C[I,J] = A[I,J] + 10 \\
 & \quad ENDO \\
 & \quad D[J] = A[N,J] \\
200: & \quad F[J] = C[10,J] \\
 & ENDO
\end{array}
$$

$$
\Rightarrow
\begin{array}{ll}
 & DO\ J = 1, M \\
 & \quad lm = TRUE \\
 & \quad DO\ I = 1, N \\
 & \quad\quad IF(lm)\ \ A[I,J] = B[I,J] + 10 \\
 & \quad\quad IF(lm)\ \ m1 = \ !L[I,J] \\
 & \quad\quad lm = lm\ \ AND\ \ m1 \\
 & \quad\quad IF(lm)\ \ C[I,J] = A[I,J] + 10 \\
 & \quad ENDO \\
 & \quad m2 = lm \\
 & \quad IF(m2)\ \ D[J] = A[N,J] \\
200: & \quad F[J] = C[10,J] \\
 & ENDO
\end{array}
$$

Exit branch

Terminates one or more loops by transferring control to a target outside a loop. E.g.

$$
\begin{array}{ll}
 & DO\ J = 1, M \\
 & \quad DO\ I = 1, N \\
 & \quad\quad A[I,J] = B[I,J] + 10 \\
S: & \quad\quad IF(L[I,J])\ \ GOTO\ 200 \\
 & \quad\quad C[I,J] = A[I,J] + 10 \\
 & \quad ENDO \\
 & \quad D[J] = A[N,J] \\
200: & \quad F[J] = C[10,J] \\
 & ENDO
\end{array}
$$

$$
\Rightarrow
\begin{array}{ll}
 & DO\ J = 1, M \\
 & \quad lm = TRUE \\
 & \quad DO\ I = 1, N \\
 & \quad\quad IF(lm)\ \ A[I,J] = B[I,J] + 10 \\
 & \quad\quad IF(lm)\ \ m1 = \ !L[I,J] \\
 & \quad\quad lm = lm\ \ AND\ \ m1 \\
 & \quad\quad IF(lm)\ \ C[I,J] = A[I,J] + 10 \\
 & \quad ENDO \\
 & \quad m2 = lm \\
 & \quad IF(m2)\ \ D[J] = A[N,J] \\
200: & \quad F[J] = C[10,J] \\
 & ENDO
\end{array}
$$

Backward branch

Transfer control to a statement occurring before the branch but at the same loop nesting level. E.g.

$$
\begin{array}{ll}
 & If(P)\ \ GOTO\ 200 \\
 & \dots \\
100: & S1 \\
 & \dots \\
200: & S2 \\
 & \dots \\
 & If(Q)\ \ GOTO\ 100
\end{array}
$$

If this code is transformed using only forward if-conversion

$$\Downarrow$$

$$
\begin{array}{ll}
 & m1 = !P \\
 & \dots \\
100; & If(m1)\ \ S1 \\
 & \dots \\
200: & S2 \\
 & \dots \\
 & If(Q)\ \ GOTO\ 100
\end{array}
$$

Backward branch

Transfer control to a statement occurring before the branch but at the same loop nesting level. E.g.

$If(P)\ \ GOTO\ 200$
...
100 : $S1$
...
200 : $S2$
...
$If(Q)\ \ GOTO\ 100$

If this code is transformed using only forward if-conversion

$\Downarrow$

$m1 = !P$
...
100; $If(m1)\ \ S1$
...
200 : $S2$
...
$If(Q)\ \ GOTO\ 100$

Backward branch

Transfer control to a statement occurring before the branch but at the same loop nesting level. E.g.

$$
\begin{array}{ll}
 & If(P)\ \ GOTO\ 200 \\
 & \dots \\
100: & S1 \\
 & \dots \\
200: & S2 \\
 & \dots \\
 & If(Q)\ \ GOTO\ 100
\end{array}
$$

If this code is transformed using only forward if-conversion

$$
\begin{array}{ll}
 & \Downarrow \\
 & m1 = \ !P \\
 & \dots \\
100; & If(m1)\ \ S1 \\
 & \dots \\
200: & S2 \\
 & \dots \\
 & If(Q)\ \ GOTO\ 100
\end{array}
$$

Backward branch cont..

Consider the same example again

$If(P)\ \ GOTO\ 200$
...
100 : $S1$
...
200 : $S2$
...
$If(Q)\ \ GOTO\ 100$

Correct transformation

$\Downarrow$

$m = P$
...
$bb = FALSE$
100; $If(!m\ \ OR\ \ bb)\ \ S1$
...
200 : $S2$
...
$If(Q)\ \ THEN$
$bb = TRUE$
$GOTO\ 100$
$ENDIF$

Backward branch cont..

Consider the same example again

$$
\begin{array}{ll}
 & If(P)\ \ GOTO\ 200 \\
 & \ldots \\
100: & S1 \\
 & \ldots \\
200: & S2 \\
 & \ldots \\
 & If(Q)\ \ GOTO\ 100
\end{array}
$$

Correct transformation

$$\Downarrow$$

$$
\begin{array}{ll}
 & m = P \\
 & \ldots \\
 & bb = FALSE \\
100; & If(!m\ \ OR\ \ bb)\ \ S1 \\
 & \ldots \\
200: & S2 \\
 & \ldots \\
 & If(Q)\ \ THEN \\
 & \quad bb = TRUE \\
 & \quad GOTO\ 100 \\
 & ENDIF
\end{array}
$$

Putting it all together

For *complete* If-Conversion:

- relocate exit branches (recursively)
- remove forward branches (while handling effects of backward branches)
- recurse for nested *DO* loops

Improvements

After removing all branches, further improvements can be made

Simplification

The boolean conditions of the guard statements can be simplified

Iterative Dependence

Control dependences which are not covered by If-conversion must be handled

If-Reconstruction

In case the code resulting from If-conversion cannot be vectorized, apply an inverse transformation to avoid overhead

References I


Ken Kennedy and Randy Allen.
Optimizing Compilers for Modern Architectures : A Dependence-Based Approach.
Morgan Kaufmann, 2001.