

Handling Control Flow

Chapter 7 of Allen & Kennedy: *Optimizing Compilers for Modern Architectures*

CSC2/455
Fall 2004

Lecturer: Yutao Zhong

Motivational example

```
DO 100 I = 1, N
S1   IF (A(I-1).GT. 0.0) GO TO 100
S2   A(I) = A(I) + B(I)*C
100 CONTINUE
```

❖ Considering only data dependences...

❖ We can vectorize this ! ...?

```
S2 A(1:N) = A(1:N) + B(1:N)*C
DO 100 I = 1, N
S1   IF (A(I-1).GT. 0.0) GO TO 100
100 CONTINUE
```

What is wrong?

URCS

CS2/455 2004

2

Motivational example

```
DO 100 I = 1, N
S1   IF (A(I-1).GT. 0.0) GO TO 100
S2   A(I) = A(I) + B(I)*C
100 CONTINUE
```

❖ We are missing dependences:

❖ a control dependence between S₁ and S₂

❖ The missing dependence completes a cycle that prevents us from applying vectorization

URCS

CS2/455 2004

3

Control dependence: definition

❖ A node x in directed graph G with a single exit node *postdominates* node y in G if any path from y to the exit node of G must pass through x.

❖ A statement y is said to be *control dependent* on another statement x if:

❖ there exists a non-trivial path from x to y such that every statement z ≠ x in the path is postdominated by y and

❖ x is not postdominated by y.

URCS

CS2/455 2004

4

Control dependence: definition

❖ Intuition behind:

❖ x must be a conditional branch

❖ y must be executed along one direction of branch x, but not both

❖ Control dependence as being a property of basic blocks

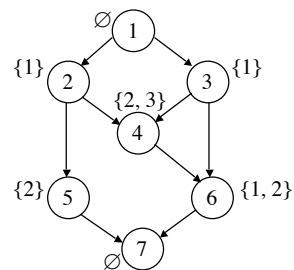
❖ no control flow change within a basic block

URCS

CS2/455 2004

5

Control dependence: example



URCS

CS2/455 2004

6

Control dependence in loops

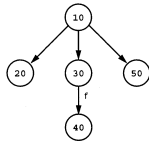
❖ A special representation:

- ❖ a loop control node to evaluate the loop control expression

❖ example

```

10 DO I = 1, 100
20  A(I) = A(I) + B(I)
30  IF (A(I) .GT. 0) GO TO 50
40  A(I) = -A(I)
50  B(I) = A(I) + C(I)
    ENDDO
    
```



URCS

CS2/455 2004

7

Old execution model

- ❖ In Chapter 2, we think of the program execution as a collection of statement instances $S(i)$, where i is the iteration vector corresponding to the nest of loops containing S .
- ❖ $S(i)$ can be executed at any time after all statement instances it depends on have been executed.
- ❖ But control dependences introduce the extra possibility ...

```

DO I = 1, N
  IF (P) GO TO 100
  S1
  S2
ENDDO
100
    
```

URCS

CS2/455 2004

8

New execution model (I)

- ❖ Add a *doit* flag for each S
- ❖ For any statement that does not control depend on any other statements, set $S.doit = True$
- ❖ For all other statements, initialize $S.doit = False$
- ❖ S can be executed only when $S.doit = True$ and all statements it depends on either has a false *doit* flag or has been executed

URCS

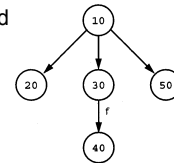
CS2/455 2004

9

New execution model (II)

- ❖ When do we set $S.doit$ to be True?

- ❖ after we evaluate a condition
- ❖ set all statements that are control dependent on this condition and whose execution is forced by the resulting sense of the condition



URCS

CS2/455 2004

10

Control dependences & transformations

- ❖ We only consider branches which generate loop-independent dependences here
- ❖ Many transformations do not affect loop-independent dependences
- ❖ Focus our attention on loop distribution (basic vectorization/parallelization)

URCS

CS2/455 2004

11

Loop distribution

```

DO I = 1, N
  S1 IF (A(I) .LT. B(I)) GOTO 20
  S2 B(I) = B(I) + C(I)
  20 CONTINUE
ENDDO
    
```

- ❖ After distributing I-loop:

```

DO I = 1, N
  S1 IF (A(I) .LT. B(I)) GOTO 20
  ENDDO
DO I = 1, N
  S2 B(I) = B(I) + C(I)
  ENDDO
20 CONTINUE
    
```

Wrong!

URCS

CS2/455 2004

12

Loop distribution

❖ Problem: control dependences may cross between distributed loops; we need a mechanism to capture the *doit* flag

❖ Solution: save the results of a conditional computation

```
DO I = 1, N
  IF (A(I).LT.B(I)) GOTO 20
  S1 B(I) = B(I) + C(I)
  S2 CONTINUE
20 ENDDO
```

❖ After transformation:

```
DO I = 1, N
  DO I = 1, N
    e(I) = A(I).LT.B(I)
  ENDDO
  DO I = 1, N
    S2 IF (e(I).EQ..FALSE.) B(I) = B(I) + C(I)
  ENDDO
```

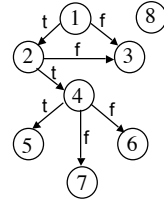
URCS

CS2/455 2004

13

A complex example (I)

```
DO I = 1, N
  IF (A(I).NE.0) THEN
  1 IF (B(I)/A(I).GT.1) GOTO 4
  2 ENDF
  A(I) = B(I)
  GOTO 8
  3 IF (A(I).GT.T) THEN
  4 T = (B(I) - A(I)) + T
  5 ELSE
  6 T = (T + B(I)) - A(I)
  7 B(I) = A(I)
  ENDF
  8 C(I) = B(I) + C(I)
ENDDO
```



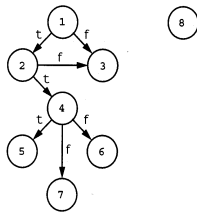
URCS

CS2/455 2004

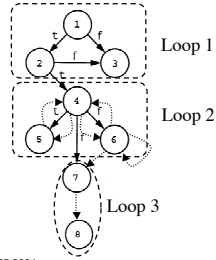
14

A complex example (II)

❖ control dependence graph



❖ control and data dependence graph



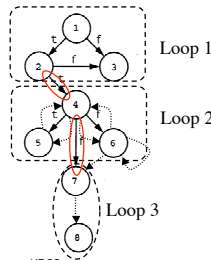
URCS

CS2/455 2004

15

A complex example (III)

❖ control and data dependence graph



❖ Need two execution variables (arrays) E2 and E4 to hold the result of branches at statement 2 and 4

❖ Consider statement 2, one of the following 3 cases must hold:

- 2 is executed and the truth branch to 4 is taken;
- 2 is executed and the false branch to 3 is taken;
- 2 is never executed because the false branch of 1 is taken

❖ Correspondingly, three values are needed for an execution variable: *true*, *false*, *undefined*

URCS

CS2/455 2004

16

A complex example (IV)

```
PARALLEL DO I = 1, N
  E2(I) = .UNDEFINED.
  1 IF (A(I).NE.0) THEN
  2 E2(I) = (B(I)/A(I).GT.1)
  ENDF
  3 IF (E2(I).NE..TRUE.) A(I) = B(I)
ENDDO
DO I = 1, N
  E4(I) = .UNDEFINED.
  IF (E2(I).EQ..TRUE.) THEN
  E4(I) = (A(I).GT.T)
  4 IF (E4(I).EQ..TRUE.) THEN
  5 T = (B(I) - A(I)) + T
  ELSE
  6 T = (T + B(I)) - A(I)
  ENDF
  ENDF
ENDDO
```

URCS

CS2/455 2004

17

Execution variables

❖ Using execution variables corresponds to setting the *doit* flag in the execution model

- a control dependence exists from some S_0 to S
- S_0 has its *doit* flag set
- the value of the conditional expression is the same as the label on the branch to S

URCS

CS2/455 2004

18

Dealing with control flow

- ❖ So far, explicit approach
 - ✦ expose as control dependences
 - ✦ apply loop distribution/fusion on dependence and data graph
 - ✦ patch up "broken" control dependences by inserting execution variables to save the results of a conditional computation until they are needed later
- ❖ An alternative approach: if-conversion
 - ✦ convert control dependences into data dependences

URCS

CS2/455 2004

19

If-conversion

- ❖ Assumption: *guarded execution* available
- ❖ Basic idea: remove branches by replacing statements affected by branches with equivalent conditionally executed statements

```
DO 100 I = 1, N
S1   IF (A(I-1) .GT. 0.0) GO TO 100
S2   A(I) = A(I) + B(I)*C
100 CONTINUE

DO I = 1, N
      IF (A(I-1) .LE. 0.0) A(I) = A(I) + B(I)*C
ENDDO
```

URCS

CS2/455 2004

20

If-conversion: one more example

```
DO 100 I = 1, N
S1   IF (A(I-1) .GT. 0.0) GO TO 100
S2   A(I) = A(I) + B(I) * C
S3   B(I) = B(I) + A(I)
100CONTINUE
```

- ❖ After if-conversion:

```
DO 100 I = 1, N
S2   IF (A(I-1) .LE. 0.0) A(I) = A(I) + B(I) * C
S3   IF (A(I-1) .LE. 0.0) B(I) = B(I) + A(I)
100CONTINUE
```

conditional assignment

- ❖ Vectorization:

```
DO 100 I = 1, N
S2   IF (A(I-1) .LE. 0.0) A(I) = A(I) + B(I) * C
100CONTINUE
S3   WHERE (A(0:N-1) .LE. 0.0) B(1:N) = B(1:N) + A(1:N)
```

URCS

CS2/455 2004

21

Guarded statements

- ❖ Statements implicitly contains a logical expression controlling its execution
- ❖ Guarded execution translates naturally into WHERE statements (FORTRAN 90) as long as data dependences permit

URCS

CS2/455 2004

22

Branch classification

- ❖ **Forward Branch:** transfers control to a target that occurs lexically after the branch but at the same level of nesting
- ❖ **Backward Branch:** transfers control to a statement occurring lexically before the branch but at the same level of nesting
- ❖ **Exit Branch:** terminates one or more loops by transferring control to a target outside a loop nest

URCS

CS2/455 2004

23

Removal of forward branches

- ❖ Simple version algorithm:
 - ✦ sweep through a program
 - ✦ maintain a Boolean expression *C* to represent the logical condition that must be true for the current statement to be executed
 - ✦ when a branch is encountered, *conjoin* its controlling expression into *C*
 - ✦ when a branch target is encountered, *disjoin* its controlling expression into *C*

URCS

CS2/455 2004

24

Removal of forward branches: example

```
DO I = 1,N
  C1 IF (A(I).GT.10) GO TO 60
  20 A(I) = A(I) + 10
  C2 IF (B(I).GT.10) GO TO 80
  40 B(I) = B(I) + 10
  60 A(I) = B(I) + A(I)
  80 B(I) = A(I) - 5
ENDDO
```

↓

```
DO I = 1,N
  m1 = A(I).GT.10
  20 IF(.NOT.m1) A(I) = A(I) + 10
  IF(.NOT.m1) m2 = B(I).GT.10
  40 IF(.NOT.m1.AND..NOT.m2) B(I) = B(I) + 10
  60 IF(.NOT.m1.AND..NOT.m2.OR.m1) A(I) = B(I) + A(I)
  80 IF(.NOT.m1.AND..NOT.m2.OR.m1.OR..NOT.m1
  .AND.m2) B(I) = A(I) - 5
ENDDO
```

URCS

CS2/455 2004

25

Removal of forward branches: example

❖ After simplification:

```
DO 100 I = 1,N
  m1 = A(I).GT.10
  20 IF(.NOT.m1) A(I) = A(I) + 10
  IF(.NOT.m1) m2 = B(I).GT.10
  40 IF(.NOT.m1.AND..NOT.m2) B(I) = B(I) + 10
  60 IF(m1.OR..NOT.m2) A(I) = B(I) + A(I)
  80 B(I) = A(I) - 5
ENDDO
```

❖ After vectorization

```
m1(1:N) = A(1:N).GT.10
20 WHERE(.NOT.m1(1:N)) A(1:N) = A(1:N) + 10
WHERE(.NOT.m1(1:N)) m2(1:N) = B(1:N).GT.10
40 WHERE(.NOT.m1(1:N).AND..NOT.m2(1:N))
  B(1:N) = B(1:N) + 10
60 WHERE(m1(1:N).OR..NOT.m2(1:N))
  A(1:N) = B(1:N) + A(1:N)
80 B(1:N) = A(1:N) - 5
```

URCS

CS2/455 2004

26

Remaining issues

- ❖ Forward branch
 - ☞ guard expression simplification
- ❖ Backward branch
 - ☞ implicit loops
 - ☞ complicating forward branch processing
- ❖ Exit branch
 - ☞ branch relocation: convert any exit branch into either a forward or a backward branch
- ❖ Iterative dependence
 - ☞ the notion of range

URCS

CS2/455 2004

27

If-conversion: side effects

- ❖ If-conversion unnecessarily complicates the code when vectorization is not possible

```
DO 100 I = 1, N
  IF (A(I) .GT. 0) GOTO 100
  B(I) = A(I) * 2.0
  A(I+1) = B(I) + 1
100 CONTINUE
```

- ❖ After if-conversion:

```
DO 100 I = 1, N
  m1 = (A(I) .GT. 0)
  IF (.NOT. m1) B(I) = A(I) * 2.0
  IF (.NOT. m1) A(I+1) = B(I) + 1
100 CONTINUE
```

- ❖ If-reconstruction needed

URCS

CS2/455 2004

28

Summary (I)

- ❖ If-conversion
 - ☞ assuming guarded execution
 - ☞ converting control dependence to data dependence
 - ☞ effective in vectorization
 - ☞ substantial useless work when vectorization is not possible

URCS

CS2/455 2004

29

Summary (II)

- ❖ Control dependence
 - ☞ explicitly exposing control dependence
 - ☞ loop distribution/fusion first, dealing with “broken” control dependences afterwards
 - ☞ execution variables: an analogy of if-conversion
 - ☞ complicated code generation

URCS

CS2/455 2004

30