Optimizing Compilers for Modern Architectures:

Handling Control Flow

Vendula Poncová
Martin Šifra

if-conversion

= process of removing all branches from a program

- branch relocation
- branch removal

branch classification

```
1  DO 100 I = 1,N
2    IF (A(I-1).GT.0.0) GOTO 4
3    A(I) = A(I) + B(I) * C
4    B(I) = B(I) + 10
5  ENDDO
```

forward branch

```
1 I = NEXT(I)
2 A(I) = A(I) + B(I)
3 IF (I.LT.1000) GOTO 1
```

backward branch

```
1 DO I = 1,N
2   IF (ABS(A(I)-B(I)).LE.DEL) GOTO 4
3 ENDDO
4 CONTINUE
```

exit branch

forward branches

```
1 DO I = 1, N
2   IF (C1) GOTO 5
3   S1                       #!C1
4   IF (C2) GOTO 6           #!C1 and !C2
5   S2                       #(!C1 and !C2) or C1
6   S3                       #(!C1 and !C2) or C1 or C2
7 ENDDO
```

```
1 DO I = 1, N
2   m1 = C1
3   IF (!m1) S1
4   IF (!m1) m2 = C2
5   IF ((!m1 and !m2) or m1) S2
6   IF ((!m1 and !m2) or m1 or m2) S3
7 ENDDO
```

- branch removal
- guarded notation

exit branches

```
1 DO J = 1, M
2   DO I = 1, N
3     S1                           #while !C1
4     IF (C1) GOTO 9
5     S2                           #while !C1
6   ENDDO
7   S3
9 ENDDO
```

```
1 DO J = 1, M
2   G = TRUE
3   DO I = 1, N
4     IF (G) S1
5     IF (G) m1 = !C1
6     G = G and m1
7     IF (G) S2
8   ENDDO
9   IF(!G) GOTO 11
10  S3
11 ENDDO
```

- branch relocation
- transformation to forward branches

backward branches

- implicit loops
- forward branches may jump into into these loops

```
1 IF (C1) GOTO 3
2 S1
3 S2
4 IF (C2) GOTO 2
```

- while loops

simplification

- guards are repeatedly evaluated at runtime
- Boolean simplification is NP-complete
- Quine–McCluskey algorithm
- streamlining

if-reconstruction

= reverse transformation to if-conversion

```
1 DO I = 1, N
2   IF (A(I) > 0) 5
3   B(I) = A(I) * 2
4   A(I+1) = B(I) + 1
5 CONTINUE
```

```
1 DO I = 1, N
2   m1 = A(I) > 0
3   IF(!m1) B(I) = A(I) * 2
4   IF(!m1) A(I+1) = B(I) + 1
5 CONTINUE
```

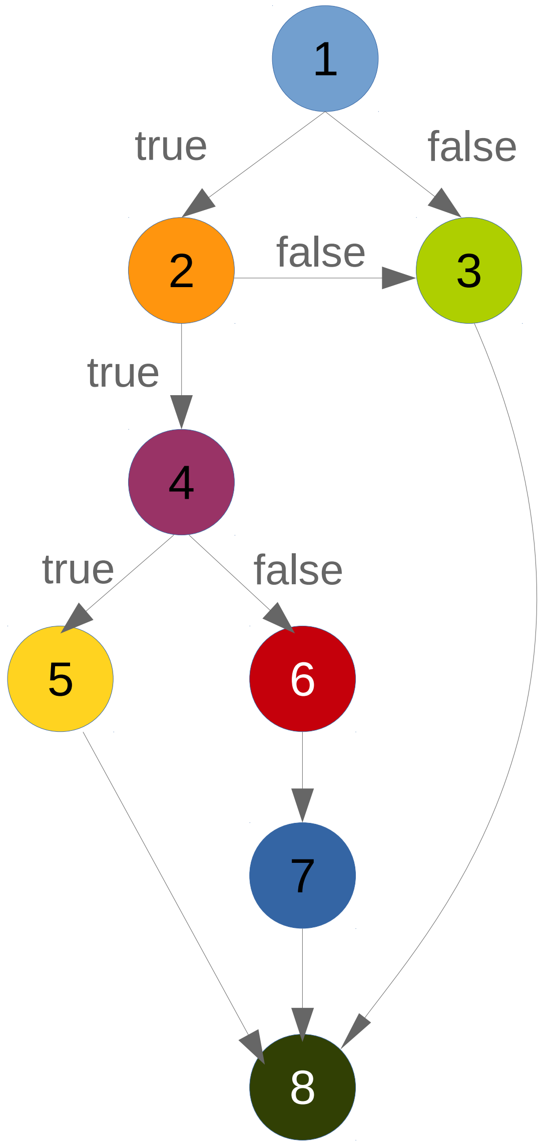
cannot be vectorized!

control dependence

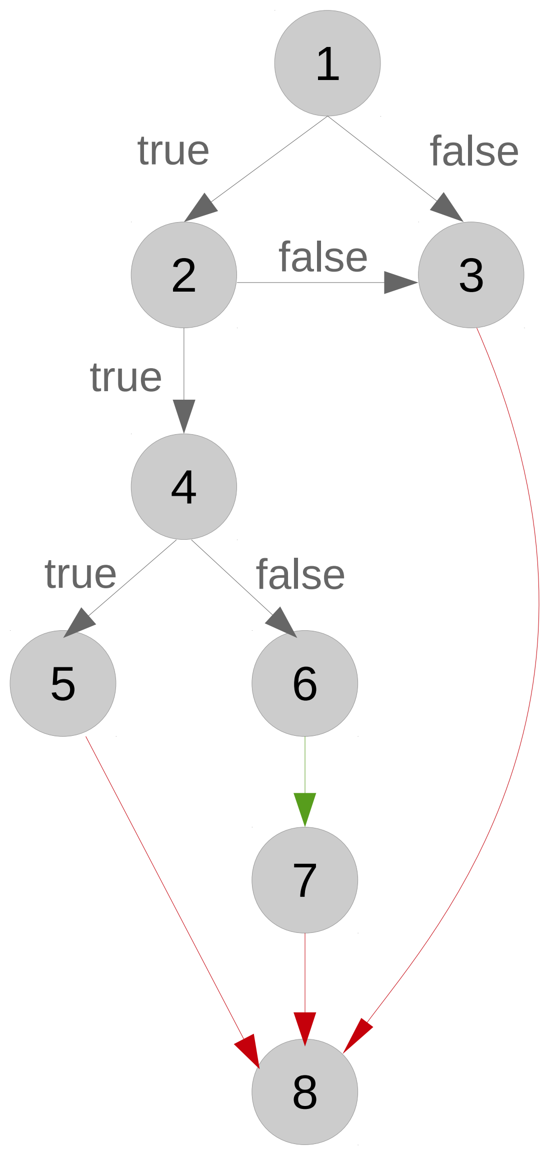
- alternative to if-conversion
- analyse the code and convert if statements only when parallelization or vectorization is possible

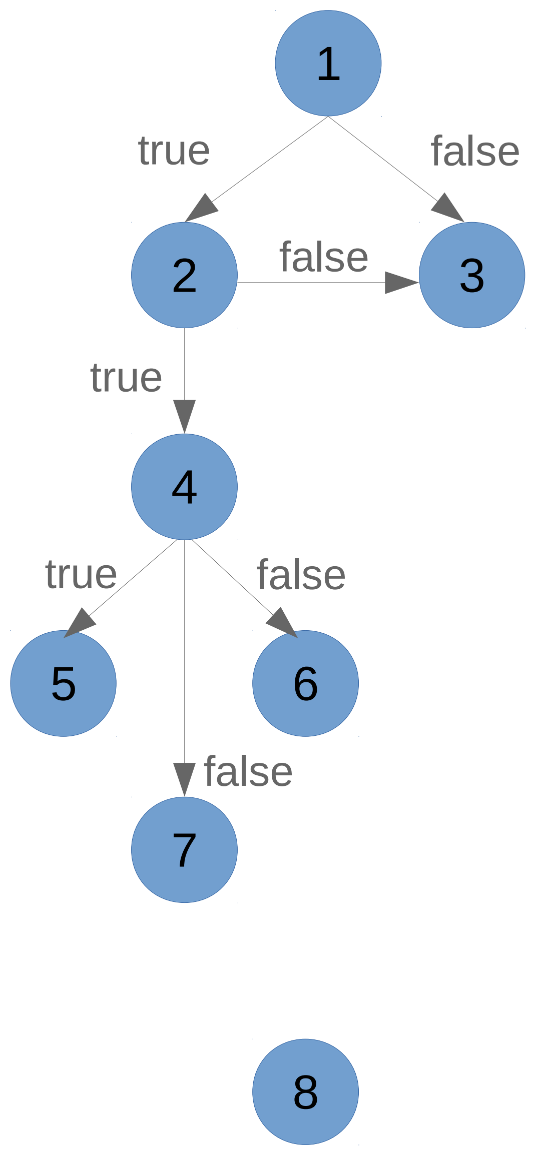
control flow graph

```
- DO I = 1, N
1    IF (C1) THEN
2      IF (C2) GOTO 4
-    ENDIF
3    S1
-    GOTO 8
4    IF (C1) THEN
5      S2
-    ELSE
6      S3
7      S4
-    ENDIF
8    S5
- ENDDO
```

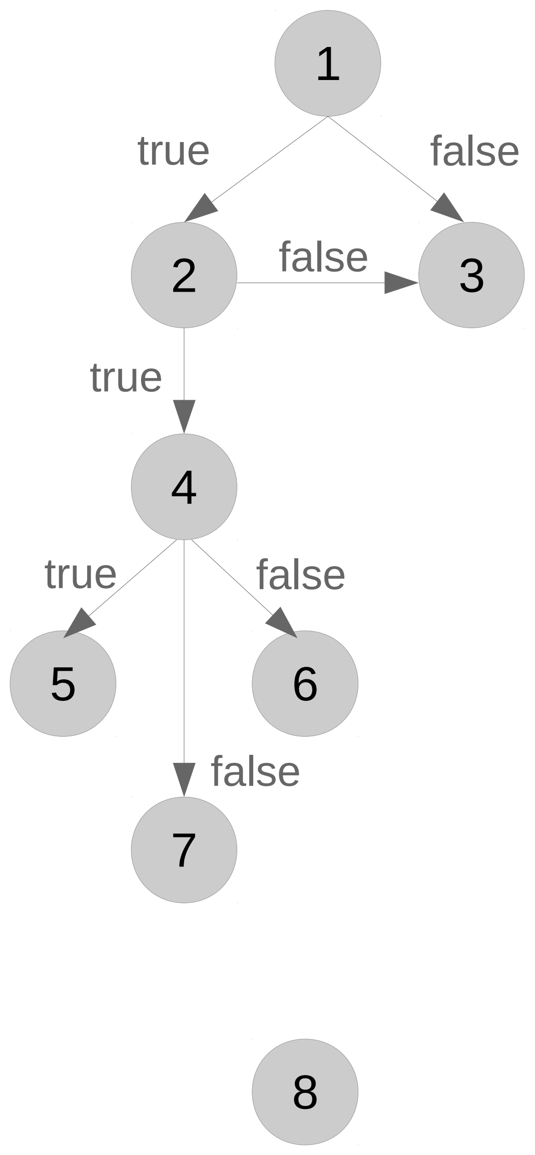


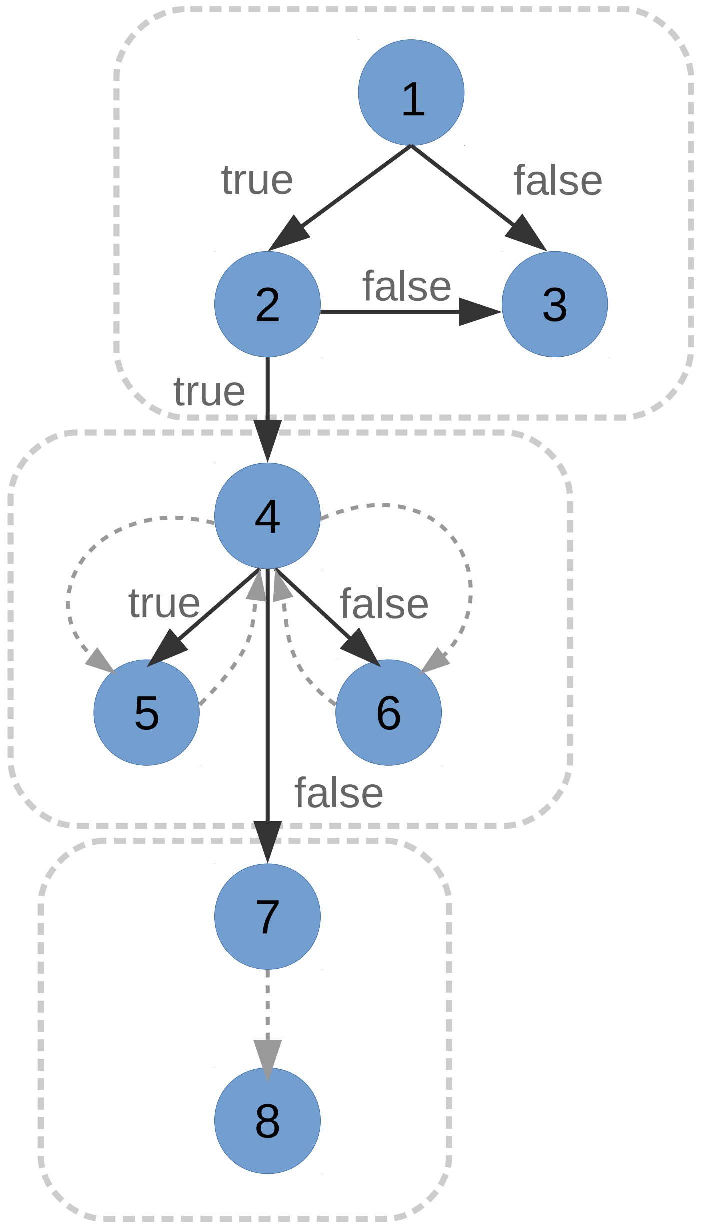
control dependence graph





control and data dependence graph





application

= how to apply control dependence graph to parallel code generation

- adapting the transformations used in code generation
- reconstruction into executable code

loop distribution

```
1 DO I = 1, N
2   IF (A(I) < B(I)) GOTO 4
3   B(I) = B(I) + C(I)
4   CONTINUE
5 ENDDO
```

```
1 DO I = 1, N
2   e(I) = A(I) < B(I)
3 ENDDO
4
5 DO I = 1, N
6   IF (!e(I)) B(I) = B(I) + C(I)
7 ENDDO
```

- similar to if-conversion

generating code

- 1. transform the control dependence graph into a set of control dependence trees
- 2. recursively generate code

summary

- if-conversion
  - eliminates all branches
  - straightforward, slow

- control dependence
  - can be used in analysis algorithms
  - complicates code generation

Questions?

title

- text