# COMP 515: Advanced Compilation for Vector and Parallel Processors

Prof. Krishna Palem
Prof. Vivek Sarkar
Department of Computer Science
Rice University
{palem,vsarkar}@rice.edu

https://wiki.rice.edu/confluence/display/PARPROG/COMP515

# Acknowledgments


- Slides from previous offerings of COMP 515 by Prof. Ken Kennedy
  - http://www.cs.rice.edu/~ken/comp515/
- POPL 1996 tutorial by Krishna Palem & Vivek Sarkar

# Control Dependences

Chapter 7

# Control Dependences

- **Constraints posed by control flow**

```
              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
```

$S_2 \; \delta_1 \; S_1$

**If we vectorize by...**

```
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
```

**…we get the wrong answer**

- **We are missing dependences**
- **There is a dependence from $S_1$ to $S_2$ - a control dependence**

# Control Dependences

- Two strategies to deal with control dependences:
  - If-conversion: expose by converting control dependences to data dependences. Used for vectorization
    - Also supported in SIMT hardware (e.g., GPGPUs) which automatically masks out statements with control conditions = false
  - Explicitly compute control dependences. Used for coarse-grained parallelism, or in cases where guarded execution is inefficient for vectorization.

# If-conversion

- **Underlying Idea: Convert statements affected by branches to 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
```

**can be converted to:**

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

# If-conversion

```
       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)
100    CONTINUE
```

- **can be converted to:**

```
        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)
100     CONTINUE
```

- **vectorize using the Fortran WHERE statement:**

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

# If-conversion

- **If-conversion assumes a target notation of guarded execution in which each statement implicitly contains a logical expression controlling its execution**

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

- **with guarded execution instead:**

```
S1   M = A(I-1).GT. 0.0
S2   IF (.NOT. M) A(I) = A(I) + B(I)*C
100 CONTINUE
```

# 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
  - The break and return statements in C are examples of exit branches, when they occur inside a loop

# If-conversion

- **If-conversion is a composition of two different transformations:**
  1. **Branch relocation**
  2. **Branch removal**

# Branch removal for If-conversion

- **Basic idea:**
  - Make a pass through the program.
  - Maintain a Boolean expression cc that represents the condition that must be true for the current expression to be executed
  - On encountering a branch, conjoin the controlling expression into cc
  - On encountering a target of a branch, its controlling expression is disjoined into cc

# Branch Removal: Forward Branches

- **Remove forward branches by inserting appropriate guards**

```
    DO 100 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 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(.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
```

# Branch Removal: Forward Branches

- **We can simplify to:**

```
  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
```

- **and then vectorize to:**

```
    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
```

# Removal of Forward Branches: Correctness

- To show correctness we must establish:
  - the guard for statement instance in the new program is true if and only if the corresponding statement in the old program is executed,
    - unless the statement has been introduced by the compiler to capture a guard variable value, which must be executed at the point the conditional expression would have been evaluated
  - the order of execution of statements in the new program with true guards is the same as the order of execution of those statements in the original program
  - Any expression with side effects is evaluated exactly as many times in the new program as in the old program

# Control Flow Graph Definition (Recap)

A *control flow graph* $CFG = (N_c, E_c, T_c)$ consists of

- $N_c$, a set of nodes. A node represents a straight-line sequence of operations with no intervening control flow i.e a basic block.

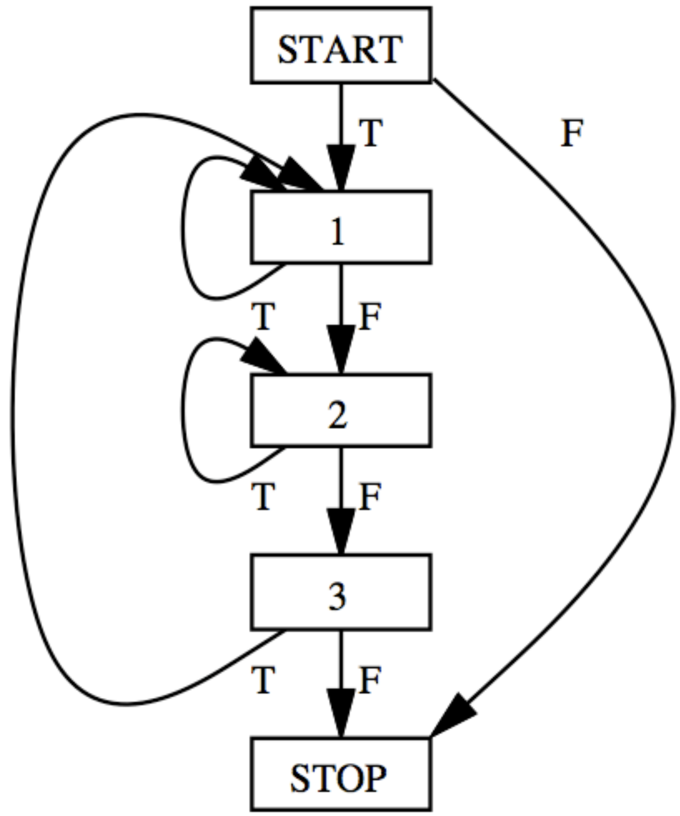
- $E_c \subseteq N_c \times N_c \times$ *Labels*, a set of *labeled* edges.

- $T_c$, a node type mapping. $T_c(n)$ identifies the type of node $n$ as one of: *START*, *STOP*, *OTHER*.

We assume that *CFG* contains a unique *START* node and a unique *STOP* node, and that for any node $N$ in *CFG*, there exist directed paths from *START* to $N$ and from $N$ to *STOP*.

# Control Flow Graph: Example

```
do {
  S1;
  if ( C1 ) continue;
  do {
    S2;
  } while ( C2 );
  S3;
} while ( C3 );
```




CONTROL FLOW GRAPH

# Workbook

```
     DO 100 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
```

1) Construct CFG
2) List basic blocks which are maximal segments without control flow
3) Write out all the data dependencies. Note: data dependencies are within a basic block.

A Predicated instruction has the following form which converts

*If(pr=true) go to s2*
*s1*

into

*If(pr=false) s1*
*If(pr=true | pr=false) s2*

```
     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(.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
```

1. Identify the predicated instructions and enumerate the relationship between predicated form and the non-predicated form.
2. Enumerate all data dependencies again?