Finals Review

CS406: Compilers

Spring 2022

Practice Exercise: CFGs and low-level loop optimizations

1. Draw CFG for the code shown. Identify loops if any. For each loop identified, mark entry node and all basic blocks.

(refer to slides 10, 36, and 48 of week12)

2. Identify loop invariant statements. Can they be moved outside their enclosing loop?

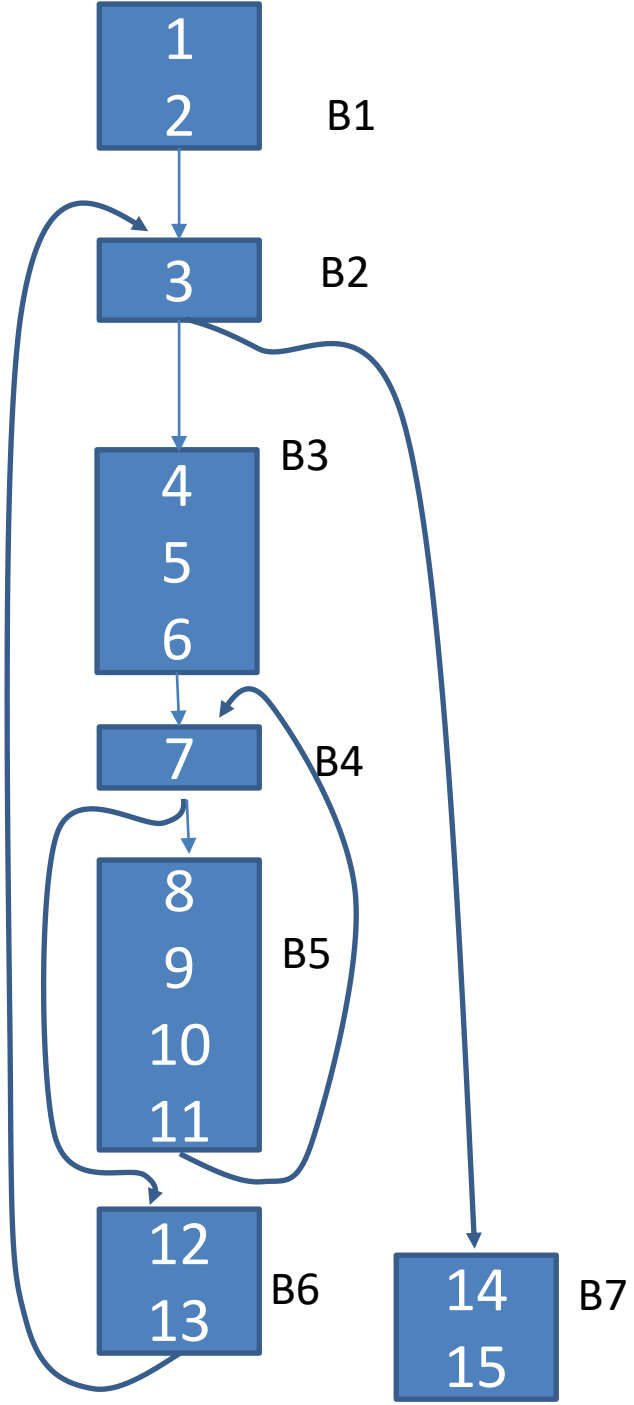
(refer to slides 59, 65-67, week12)

3. Identify induction variables. Show the code that results after performing possible strength reduction

(refer to slides 59, 65-69, week12)

```
1. X = 2;
2. Y = 10;
3. if (X < Y) goto 14
4. A = Y * X;
5. B = X * 2 + Y;
6. Z = 10;
7. if (B < Z) goto 12
8. D = Y + Z * -3;
9. Q = Y - 8;
10. Z = Z - Q;
11. goto 7;
12. X = X + 2;
13. goto 3;
14. Y = D;
15. halt;
```

```
1. X = 2;
2. Y = 10;
3. if (X < Y) goto 14
4. A = Y * X;
5. B = X * 2 + Y;
6. Z = 10;
7. if (B < Z) goto 12
8. D = Y + Z * -3;
9. Q = Y - 8;
10. Z = Z - Q;
11. goto 7;
12. X = X + 2;
13. goto 3;
14. Y = D;
15. halt;
```

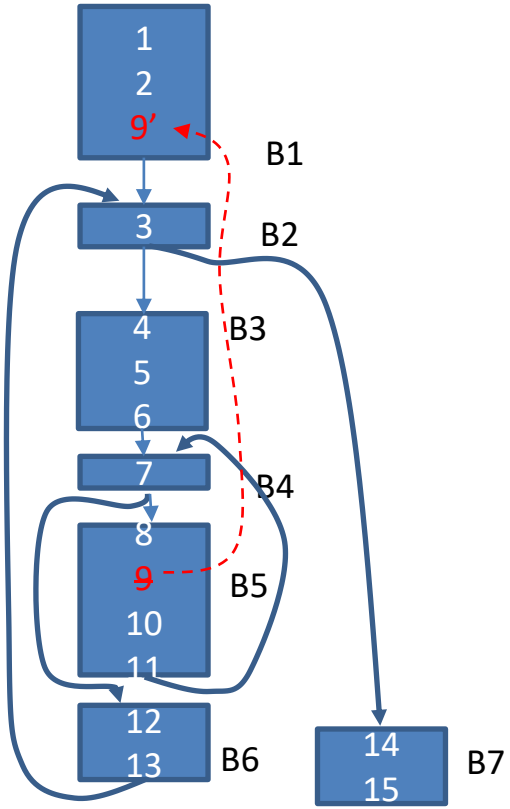


Loop headers/entry nodes: B2 and B4. The BBs in the loop for B2 are: B2, B3, B4, B5, B6. The BBs in the loop for B4 are: B4, B5.

Loop invariant statements: 6, 9.

- 6 cannot be moved, because Z is defined twice inside the loop (note that if we try to move it outside the loop, Z will not get reinitialized before the inner loop).
- 9 can be moved, though. Note that first it gets moved outside the inner loop, but then is still loop invariant (because Y is only defined outside the loop), so can be moved outside the outer loop, as well. Code after moving invariant statement:

```
1.   X = 2;
2.   Y = 10;
9'.  Q = Y - 8;
3.   if (X < Y) goto 14
4.     A = Y * X;
5.     B = X * 2 + Y;
6.     Z = 10;
7.     if (B < Z) goto 12
8.       D = Y + Z * -3;
10.      Z = Z - Q;
11.      goto 7;
12.    X = X + 2;
13.    goto 3;
14.  Y = D;
15.  halt;
```



In the inner loop, Z is an induction variable (because Q is loop invariant), and D is a mutual induction variable. After performing strength reduction on the inner loop, we get:

```
1.  X = 2;
2.  Y = 10;
9'. Q = Y - 8;
3.  if (X < Y) goto 14
4.    A = Y * X;
5.    B = X * 2 + Y;
6.    Z = 10;
8'.   D' = Y + Z * -3;
7.    if (B < Z) goto 12
8.      D = D'
10.     Z = Z - Q;
10'.    D' = D' + -3 * -Q;
11.     goto 7;
12.  X = X + 2;
13.  goto 3;
14. Y = D;
15. halt;
```

In the outer loop, X is an induction variable, and both A and B are mutual induction variables. After strength reduction, we get:

```
1.  X = 2;
2.  Y = 10;
9’. Q = Y - 8;
4’. A’ = Y * X;
5’. B’ = X * 2 + Y;
3. if (X < Y) goto 14
4.   A = A’
5.   B = B’
6.   Z = 10;
6’.  D’ = Y + Z * -3;
7.   if (B < Z) goto 12
8.      D = D’
10.     Z = Z - Q;
10’.    D’ = D’ + -3 * -Q;
11.     goto 7;
12.   X = X + 2;
12’.  A’ = A’ + 2 * Y;
12’’  B’ = B’ + 4;
13.   goto 3;
14. Y = D; 15. halt
```

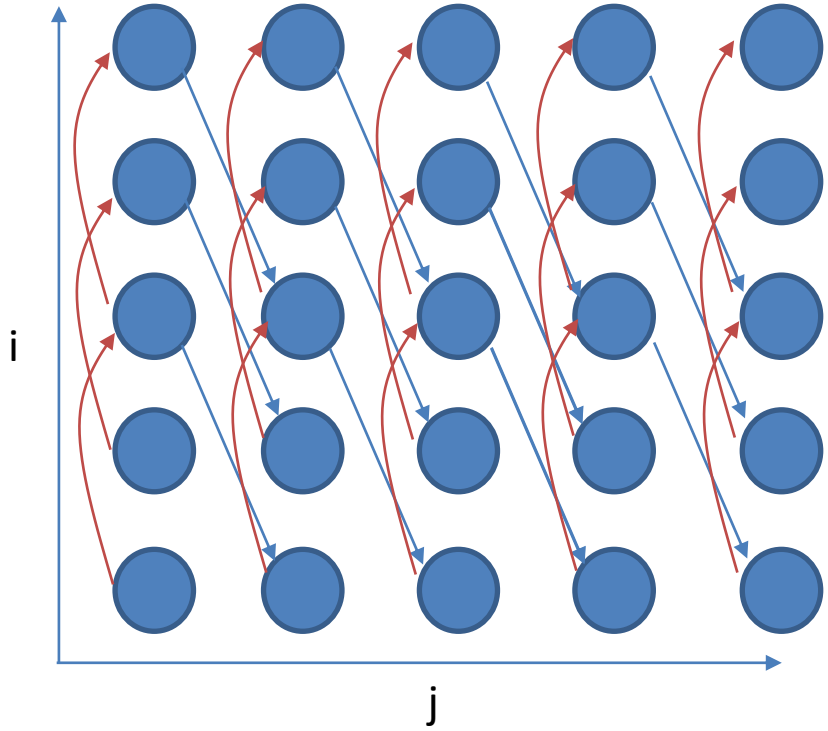
Exercise

1. Draw iteration graph for:

```
for(j=0;j<5;j++)
    for(i=0;i<5;i++)
            a[j][i]=a[j-1][i+2]+a[j][i-2]
```

2. What are the distance and direction vectors?
3. Can the loops be interchanged?
4. Repeat 1,2,and 3 for the following loop.

```
for(i=0;i<5;i++)
    for(j=0;j<5;j++)
        a[i][j]=a[i-1][j-2]
```



The blue arrows are the dependences between the write and the A[j-1][i+2]read, and the red arrows are the dependences between the write and the A[j][i-2]read.

The distance vectors are (1,–2) and (0,2)

The direction vectors are (+,–) and (0,+) (or, alternately, (<,>) and (0,<))

The loops cannot be interchanged because the (1,–2) flow dependence would trans-form into a (–2,1) dependence, which is not possible (more specifically, we would eliminate the flow dependence and introduce a (2,–1) anti-dependence).

Short Quiz

- https://forms.gle/zmUph7ixTN9CaUMF8

Practice Exercise: Peephole Optimizations - CSE

1. Show the results of performing CSE on the code shown. Write 3AC after performing CSE.

2. Suppose A and F are aliased. How would that change the results of CSE

```
1. READ(A)
2. READ(B)
3. C = A + B
4. A = A + B
5. B = C * D
6. T1 = C * D
7. T2 = T1 + C
8. F = A + B
9. C = F + B
10. G = A + B
11. T3 = F + B
12. WRITE(T3)
```

Worksheet

	Expression available before executing the stmt	After performing CSE
1. READ(A)	1.	
2. READ(B)	2.	
3. C = A + B	3.	
4. A = A + B	4.	
5. B = C * D	5.	
6. T1 = C * D	6.	
7. T2 = T1 + C	7.	
8. F = A + B	8.	
9. C = F + B	9.	
10. G = A + B	10.	
11. T3 = F + B	11.	
12. WRITE(T3)	12.	

Worksheet

Suppose A and F are aliased	Expression available before executing the stmt	After performing CSE
1. READ(A)	1.	
2. READ(B)	2.	
3. C = A + B	3.	
4. A = A + B	4.	
5. B = C * D	5.	
6. T1 = C * D	6.	
7. T2 = T1 + C	7.	
8. F = A + B	8.	
9. C = F + B	9.	
10. G = A + B	10.	
11. T3 = F + B	11.	
12. WRITE(T3)	12.	

Each row shows in parentheses what expressions are available before the expression is evaluated.

```
1. READ(A)
2. READ(B)
3. C = A + B
4. A = A + B      (A + B)
5. B = C * D      (nothing -- writing to A kills A + B)
6. T1 = C * D        (C * D)
7. T2 = T1 + C       (C * D)
8. F = A + B      (T1 + C, C * D)
9. C = F + B      (A + B, T1 + C, C * D)
10. G = A + B        (A + B, F + B)
11. T3 = F + B       (A + B, F + B)
12. WRITE(T3)        (A + B, F + B)
```

After performing CSE:

```
1. READ(A)
2. READ(B)
3. C = A + B
4. A = C          (A + B)
5. B = C * D      (nothing -- writing to A kills A + B)
6. T1 = B         (C * D)
7. T2 = T1 + C    (C * D)
8. F = A + B      (T1 + C, C * D)
9. C = F + B      (A + B, T1 + C, C * D)
10. G = F         (A + B, F + B)
11. T3 = C        (A + B, F + B)
12. WRITE(T3)     (A + B, F + B)
```

If A and F are aliased, writing to F will kill any expression that uses A (and vice versa), and also, computing F + B is the same as computing A + B.

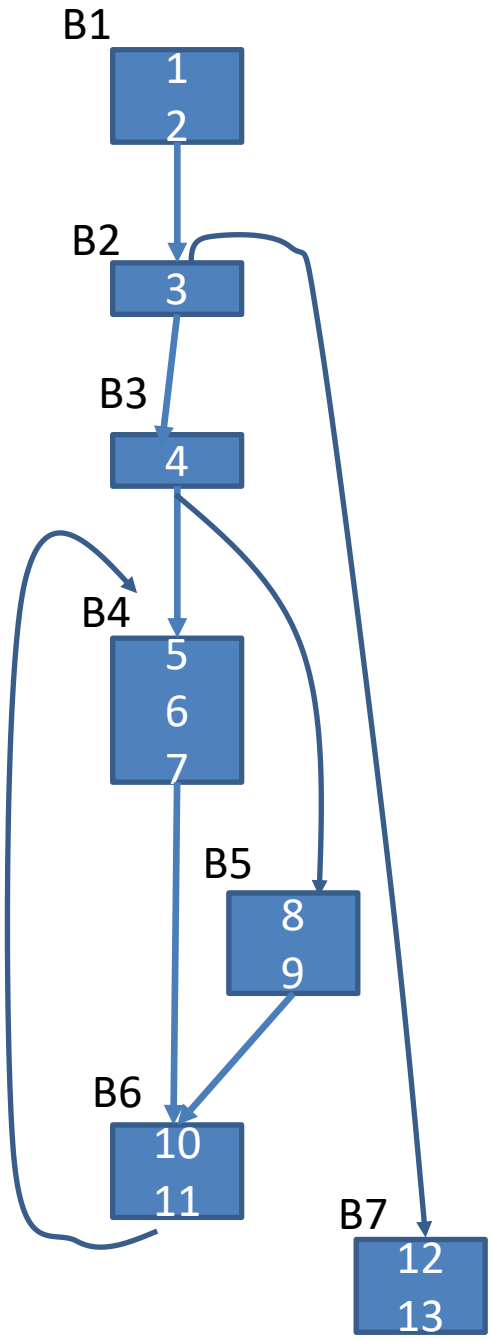
```
1. READ(A)
2. READ(B)
3. C = A + B
4. A = C          (A + B, F + B)
5. B = C * D      (nothing --writing to A kills A+B and F+B)
6. T1 = B         (C * D)
7. T2 = T1 + C    (C * D)
8. F = A + B      (T1 + C, C * D)
9. C = F + B      (T1 + C, C * D -- writing to F kills A+B)
10. G = C         (A + B, F + B – we are using C, not F)
11. T3 = C        (A + B, F + B)
12. WRITE(T3)     (A + B, F + B)
```

Practice Exercise: Dataflow analysis (available expressions)

1. Show the results of running an "available expressions" analysis on the code shown. For each line of code, show which expressions are available in that line of code.

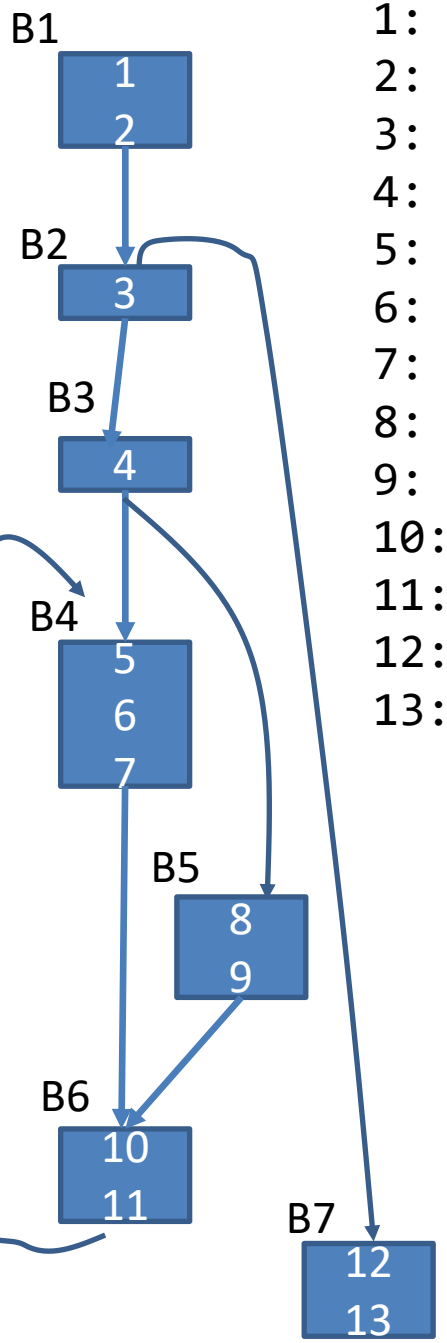
(refer to slide 8, week 14. Section 9.2.6 in Dragon book)

```
1: x = 4;
2: y = 7;
3: if (x > c) goto 12
4: if (y > 3) goto 8
5: c = x + 1;
6: b = a + x;
7: goto 10
8: a = a + x;
9: b = x + 1;
10: y = a + x;
11: goto 3;
12: c = a + x
13: halt
```




```
1: x = 4;
2: y = 7;
3: if (x > c) goto 12
4: if (y > 3) goto 8
5: c = x + 1;
6: b = a + x;
7: goto 10
8: a = a + x;
9: b = x + 1;
10: y = a + x;
11: goto 3;
12: c = a + x
13: halt
```

worksheet




```
1: x = 4;
2: y = 7;
3: if (x > c) goto 12
4: if (y > 3) goto 8
5: c = x + 1;
6: b = a + x;
7: goto 10
8: a = a + x;
9: b = x + 1;
10: y = a + x;
11: goto 3;
12: c = a + x
13: halt
```

Basic Block	Pred.	Succ.	Gen	Kill
B1	Entry	B2		
B2	B6,B1	B3,B7		
B3	B2	B4,B5		
B4	B3	B6		
B5	B3	B6		
B6	B4,B5	B2		
B7	B2	Exit		

Basic Block	Pred.	Succ.	Gen	Kill
B1	Entry	B2	4,7	x+1, a+x, x>c, y>3
B2	B6,B1	B3,B7	x>c	-
B3	B2	B4,B5	y>3	-
B4	B3	B6	x+1,a+x	x>c
B5	B3	B6	x+1	a+x
B6	B4,B5	B2	a+x	y>3
B7	B2	Exit	a+x	x>c

Basic Block	IN	OUT
B1	-	4, 7
B2	4, 7	4, 7, x>c
B3	4, 7, x>c	4, 7, x>c, y>3
B4	4, 7, x > c, y > 3	4, 7, y > 3, x + 1, a + x
B5	4, 7, x > c, y>3	4, 7, x > c, x+1, y>3
B6	4, 7, y>3, x+1, a + x	4, 7, x + 1, a + x, y>3
B7	4, 7, x > c	4, 7, a+x