

Write your answers on another sheet of paper. Homework assignments are to be completed individually. Hand written submissions are fine, but they must be readable. Due at the beginning of class. Total points: 100, 5% of course grade

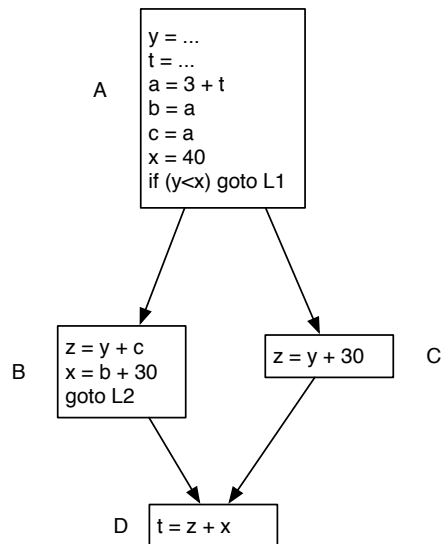
1. [30 Points] Here is an example program.

```
y = ...
t = ...
a = 3 + t
b = a
c = a
x = 40
if (y<x) goto L1
    z = y + c
    x = b + 30
    goto L2
L1:
    z = y + 30
L2:
    t = z + x
```

- (a) Draw a control-flow graph for the given program.
(b) Show the dominance frontiers.
(c) Show the SSA representation for the given program.

Answer:

(a)



(b)

Node	Dominance Frontier
A	\emptyset
B	$\{ D \}$
C	$\{ D \}$
D	\emptyset

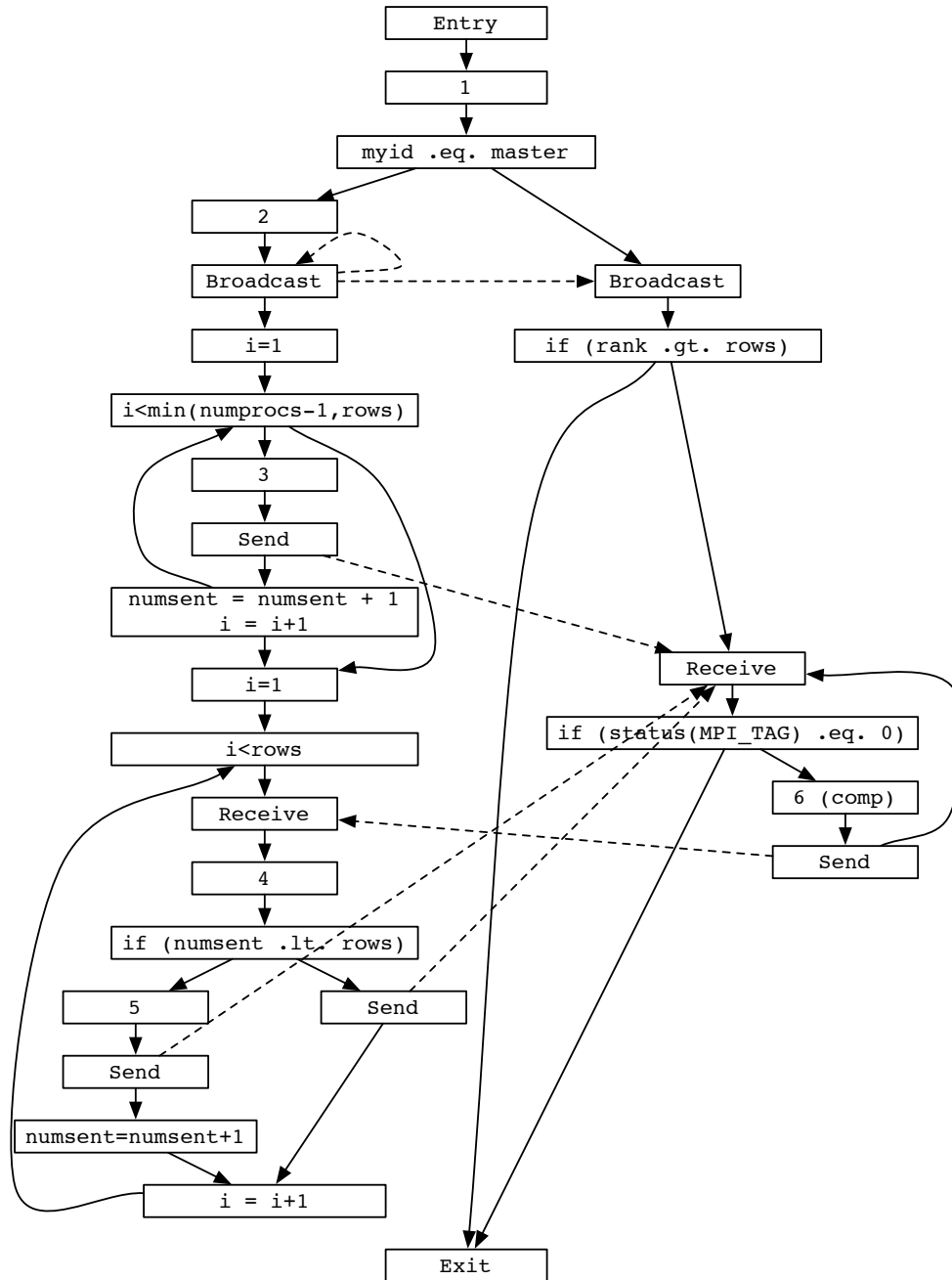
(c)

```
y0 = ...
t0 = ...
a0 = 3 + t0
b0 = a0
c0 = a0
x0 = 40
if (y0 < x0)
    z0 = y0 + c0
    x1 = b0 + 30

    z1 = y0 + 30

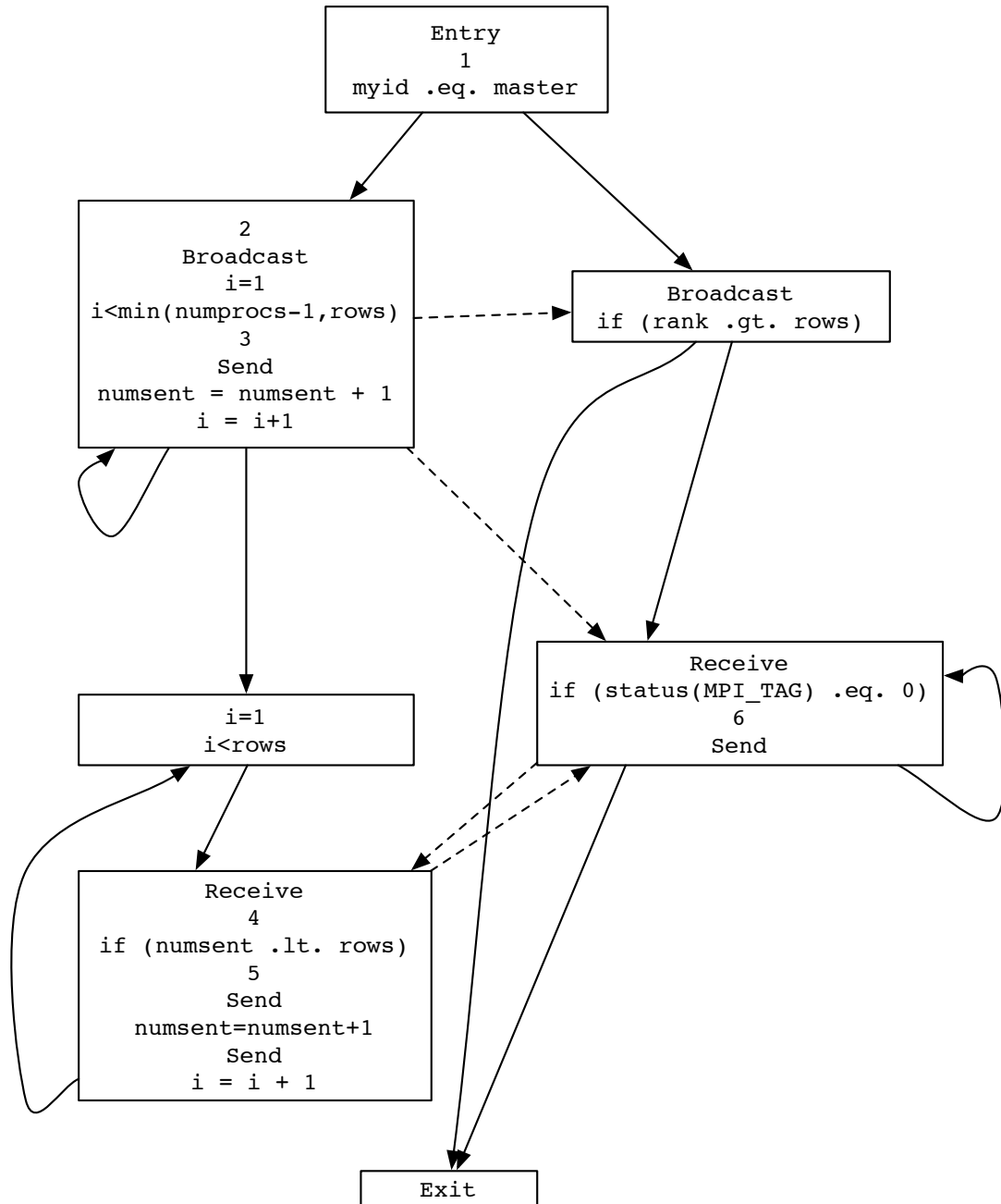
z2 = phi( z0, z1 )
x2 = phi( x1, x0 )
t1 = z2 + x2
```

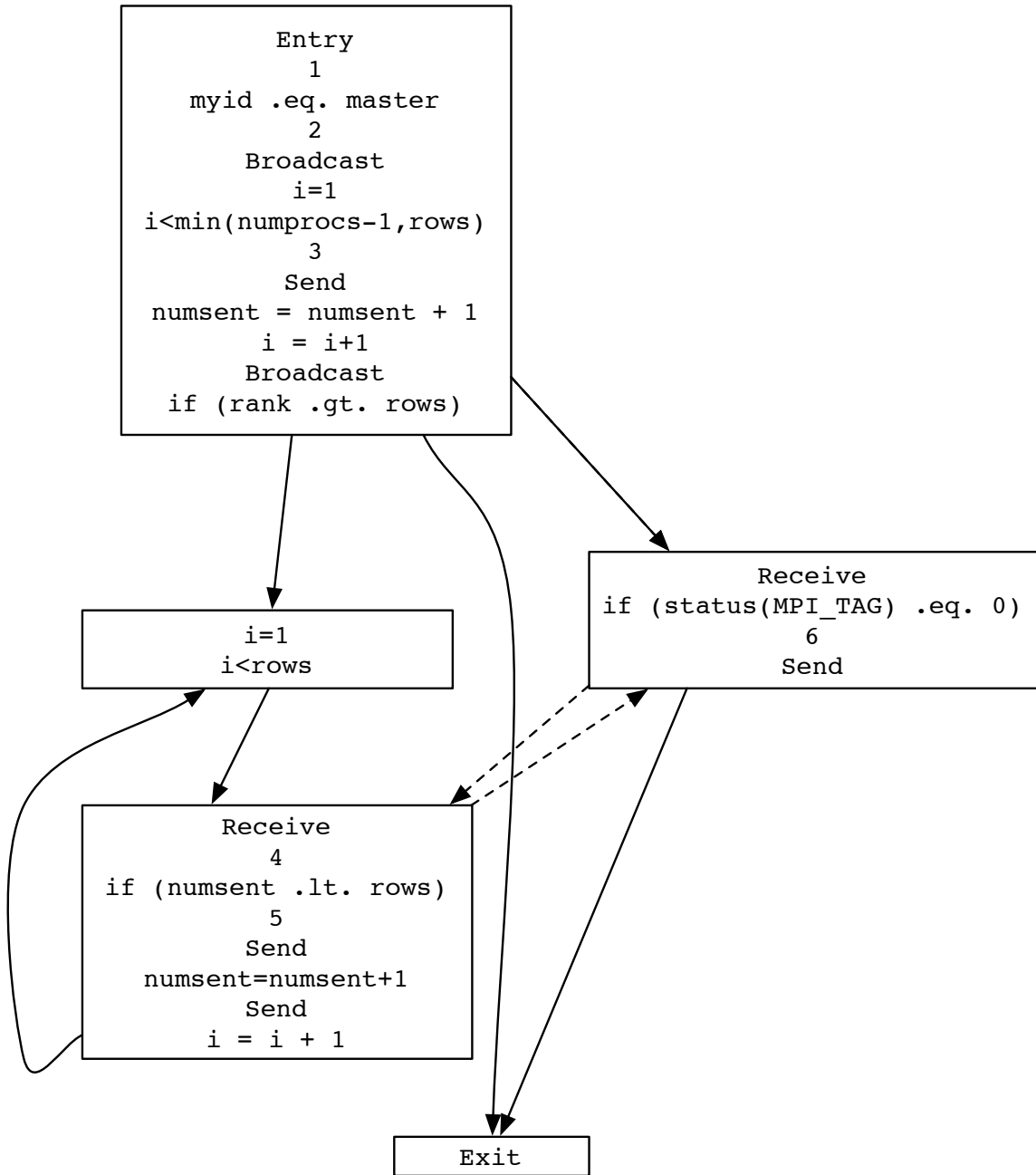
2. [20 Points] Below is the MPI-CFG for a parallel version of matrix-vector multiplication. An MPI-CFG is a control-flow graph extended with communication edges to indicate possible pairings between message passing sends and receives. The nodes labeled with just numbers include more code that has been removed since the code is not immediately relevant. If the communication edges are not treated any differently than the control-flow edges, then is the graph reducible? Explain why it is reducible or not.



Answer:

A reducible graph can be reduced to one node by applying T1 and T2 transformations to it. It is not possible to reduce the given graph. Notice that after the second set of transformations, all nodes other than the entry node have more than one predecessor.



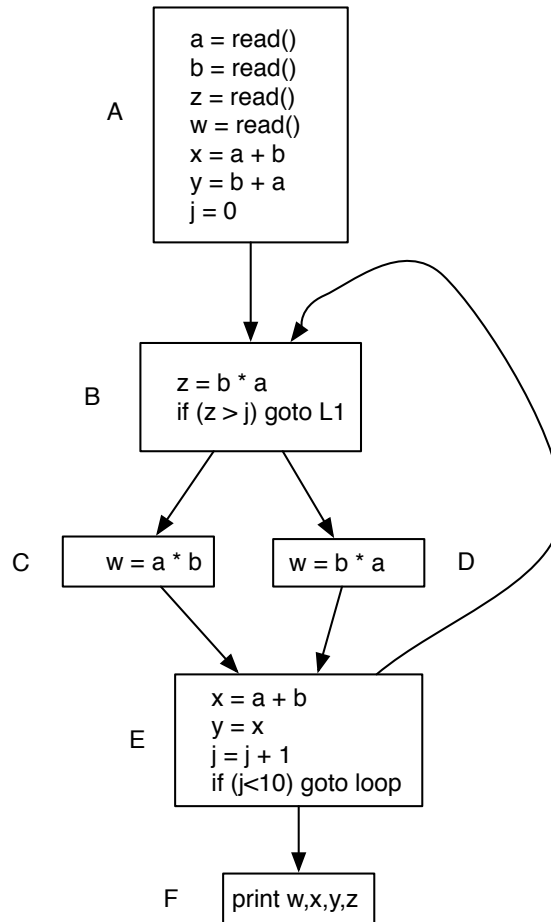


3. [50 Points] Consider the following code (assume that read is a function that reads input from stdin):

```
a = read()
b = read()
z = read()
w = read()
x = a + b
y = b + a
j = 0
loop:
  z = b * a
  if (z > j) goto L1
  w = a * b
  goto L2
L1:
  w = b * a
L2:
  x = a + b
  y = x
  j = j + 1
  if (j<10) goto loop
print w,x,y,z
```

- Compute the dominators for the given program.
- What statements are involved in loop(s)?
- Perform pessimistic global value numbering.
- Perform optimistic global value numbering.

Answer:



(a)

dom(n) is the set of nodes that dominates node n.

Node	Dominates	n	dom(n)
A	{ A, B, C, D, E, F }	A	{ }
B	{ B, C, D, E, F }	B	{ A, B }
C	{ C }	C	{ A, B, C }
D	{ D }	D	{ A, B, D }
E	{ E, F }	E	{ A, B, E }
F	{ F }	F	{ A, B, E, F }

(b) B, C, D, and E are involved in a loop. We can determine all of the nodes in a loop by tracing back from the source of the back edge until finding the sink of the back edge. The sink of the back edge is the head of the loop. All nodes found on the trace are in the loop.

(c)

For both pessimistic and optimistic global value numbering, we need to translate the code to SSA.

```
a0 = read()
b0 = read()
z0 = read()
w0 = read()
x0 = a0 + b0
y0 = b0 + a0
j0 = 0
loop:
  z1 = phi_n(z0,z2)
  x1 = phi_n(x0,x2)
  y1 = phi_n(y0,y2)
  j1 = phi_n(j0,j2)
  w1 = phi_n(w0,w4)
  z2 = b0 * a0
  if (z2 > j1) goto L1
  w3 = a0 * b0
  goto L2
L1:
  w2 = b0 * a0
L2:
  w4 = phi_m(w3,w2)
  x2 = a0 + b0
  y2 = x2
  j2 = j1 + 1
  if (j2<10) goto loop
print w4,x2,y2,z2
```

Pessimistic global value numbering.

Variable	Init Value Num	After visiting stmts
a0	#1	#1
b0	#2	#2
z0	#3	#3
w0	#4	#4
x0	#5	#1 + #2 → #20
y0	#6	#2 + #1 → #20
j0	#7	#7
z1	#8	phi_n (#3, #13) → #21
x1	#9	phi_n (#5, #17) → #22
y1	#10	phi_n (#20, #18) → #23
j1	#11	phi_n (#7, #19) → #24
w1	#12	phi_n (#4, #16) → #25
z2	#13	#2 * #1 → #26
w2	#14	#2 * #1 → #26
w3	#15	#1 * #2 → #26
w4	#16	phi_n (#26, #26) → #26
x2	#17	#1 + #2 → #20
y2	#18	#20
j2	# 19	#24 + 1 → #27

We end up with the following congruence classes:

P0 = { a0 }

P1 = { b0 }

P2 = { z0 }

P3 = { w0 }

P5 = { x0, y0, x2, y2 }

P6 = { j0 }

P7 = { z1 }

P8 = { x1 }

P9 = { y1 }

P10 = { j1 }

P11 = { w1 }

P12 = { z2, w2, w3, w4 }

P13 = { j2 }

We can rewrite the code as follows:

```

a0 = read()
b0 = read()
z0 = read()
w0 = read()
x0 = a0 + b0
y0 = x0

```

```

    j0 = 0
loop:
    z1 = phi_n(z0,z2)
    x1 = phi_n(x0,x2)
    y1 = phi_n(y0,y2)
    j1 = phi_n(j0,j2)
    w1 = phi_n(w0,w4)
    z2 = b0 * a0
    if (z2 > j1) goto L1
    w3 = z2
    goto L2
L1:
    w2 = z2
L2:
    w4 = z2
    x2 = x0
    y2 = x0
    j2 = j1 + 1
    if (j2<10) goto loop
print w4,x2,y2,z2

```

After copy propagation on SSA, which does the associated dead-code elimination at the same time, the code becomes the following:

```

    a0 = read()
    b0 = read()
    z0 = read()
    w0 = read()
    x0 = a0 + b0
    j0 = 0
loop:
    j1 = phi_n(j0,j2)
    z2 = b0 * a0
    if (z2 > j1) goto L1
    goto L2
L1:
L2:
    j2 = j1 + 1
    if (j2<10) goto loop
print z2,x0,x0,z2

```

(d) For optimistic global value numbering we start with an initial partitioning of variables based on the following rules:

- Each `read()` function can return a different value and therefore does not induce confluence.
- Any variables involved in a copy start out as confluent.
- Any variables defined with the same function (other than with `read()`) start out as confluent.

S0 = { a0 }

S1 = { b0 }

S2 = { z0 }

S3 = { w0 }

S4 = { x0, y0, x2, y2, j2 }

S5 = { j0 }

S6 = { z1, x1, y1, j1, w1 }

S7 = { z2, w3, w2 }

S8 = { w4 }

S2 properly splits S6

S9 = { z1 }

S10 = { x1, y1, j1, w1 }

S3 properly splits S10

S11 = { w1 }

S12 = { x1, y1, j1 }

S5 properly splits S12

S13 = { j1 }

S14 = { x1, y1 }

S13 properly splits S4

S15 = { j2 }

S16 = { x0, y0, x2, y2 }

The final set of congruence classes is as follows:

S0 = { a0 }

S1 = { b0 }

S2 = { z0 }

S3 = { w0 }

S5 = { j0 }

S7 = { z2, w3, w2 }

S8 = { w4 }

S9 = { z1 }

```

S11 = { w1 }
S13 = { j1 }
S14 = { x1, y1 }
S15 = { j2 }
S16 = { x0, y0, x2, y2 }

```

We can rewrite the code as follows:

```

    a0 = read()
    b0 = read()
    z0 = read()
    w0 = read()
    x0 = a0 + b0
    y0 = x0
    j0 = 0
loop:
    z1 = phi_n(z0,z2)
    x1 = phi_n(x0,x2)
    y1 = x1
    j1 = phi_n(j0,j2)
    w1 = phi_n(w0,w4)
    z2 = b0 * a0
    if (z2 > j1) goto L1
    w3 = z2
    goto L2
L1:
    w2 = z2
L2:
    w4 = phi_n( w3, w2 )
    x2 = x0
    y2 = x0
    j2 = j1 + 1
    if (j2<10) goto loop
print w4,x2,y2,z2

```

After copy propagation on SSA, which does the associated dead-code elimination at the same time, the code becomes the following:

```

    a0 = read()
    b0 = read()
    z0 = read()
    w0 = read()
    x0 = a0 + b0
    j0 = 0
loop:
    j1 = phi_n(j0,j2)

```

```
    z2 = b0 * a0
    if (z2 > j1) goto L1
    goto L2
L1:
L2:
    j2 = j1 + 1
    if (j2<10) goto loop
print z2,x0,y0,z2
```

Same as pessimistic because x1 and y1 are never used.