Introduction to Optimization[1]

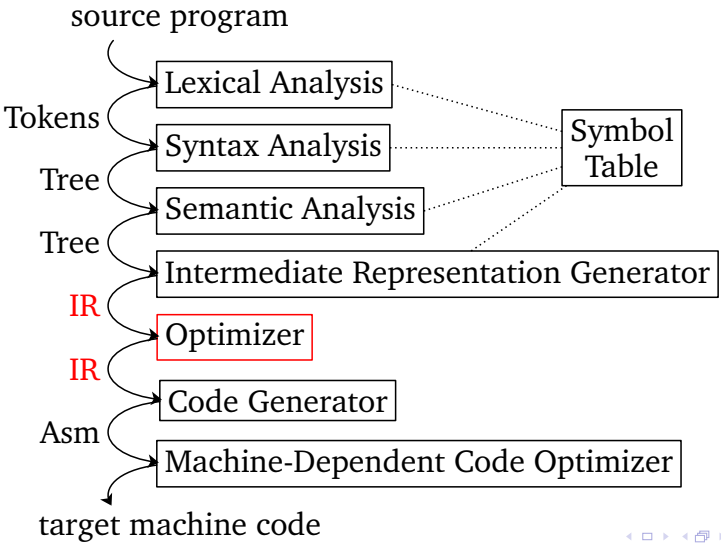
Kristoffer H. Rose
krisrose@cs.nyu.edu

Compiler Construction
CSCI-GA.2130-001/Spring 2014
NYU Courant Institute

April 28, 2014

[1] ALSU §9.1

Sixth compilation phase



Sources of Redundancy

- Programmer “cut-n-paste”
- Compiler templates.
- Insufficient state transfer.

Sources of Redundancy

- Programmer “cut-n-paste”
- Compiler templates.
- Insufficient state transfer.

Sources of Redundancy

- Programmer “cut-n-paste”
- Compiler templates.
- Insufficient state transfer.

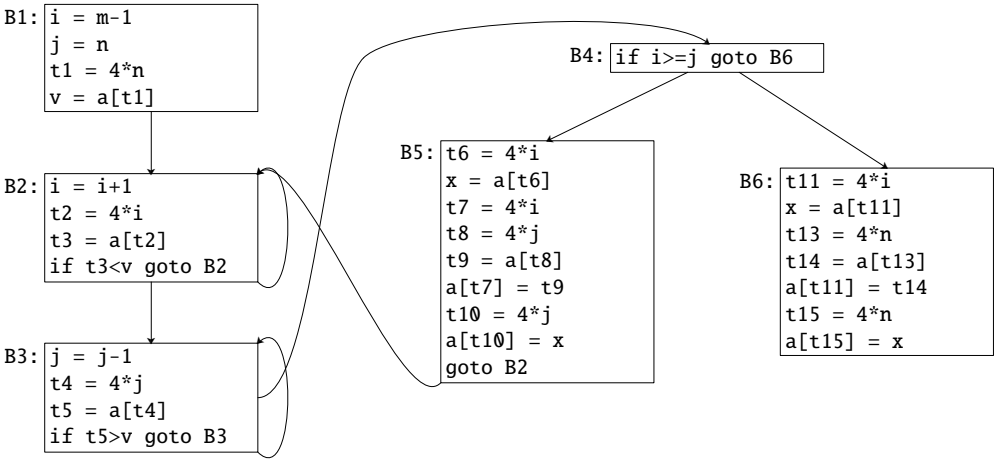
Example

```
void quicksort(int a[], int m, int n)
{
  int i, j, v, x; if (n <= m) return;

  i = m-1; j = n; v = a[n];
  while (1) {
    do i = i+1; while (a[i] < v);
    do j = j-1; while (a[j] > v);
    if (i >= j) break;
    x = a[i]; a[i] = a[j]; a[j] = x;
  }
  x = a[i]; a[i] = a[n]; a[n] = x;

  quicksort(a,m,j); quicksort(a,i+1,n);
}
```

Basic Blocks



(Local) Common Subexpression Elimination

B5:
```
t6 = 4*i
x = a[t6]
t7 = 4*i
t8 = 4*j
t9 = a[t8]
a[t7] = t9
t10 = 4*j
a[t10] = x
goto B2
```

$\Rightarrow$

B5:
```
t6 = 4*i
x = a[t6]
t8 = 4*j
t9 = a[t8]
a[t6] = t9
a[t8] = x
goto B2
```

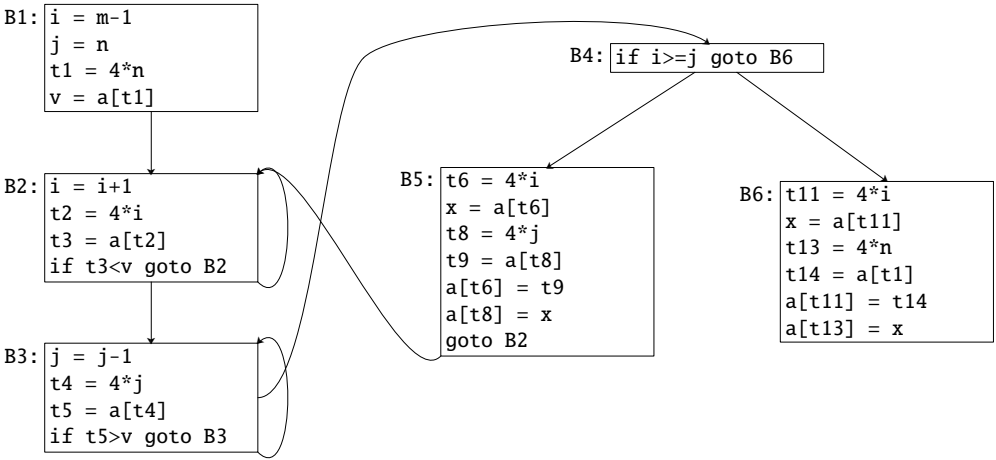
(Local) Common Subexpression Elimination

```
B5: t6 = 4*i
    x = a[t6]
    t7 = 4*i
    t8 = 4*j
    t9 = a[t8]
    a[t7] = t9
    t10 = 4*j
    a[t10] = x
    goto B2
```

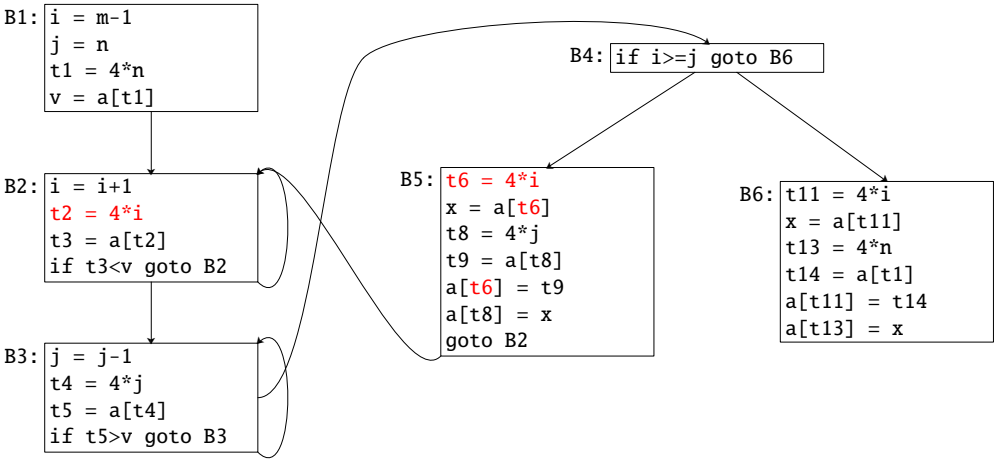
$\Rightarrow$

```
B5: t6 = 4*i
    x = a[t6]
    t8 = 4*j
    t9 = a[t8]
    a[t6] = t9
    a[t8] = x
    goto B2
```

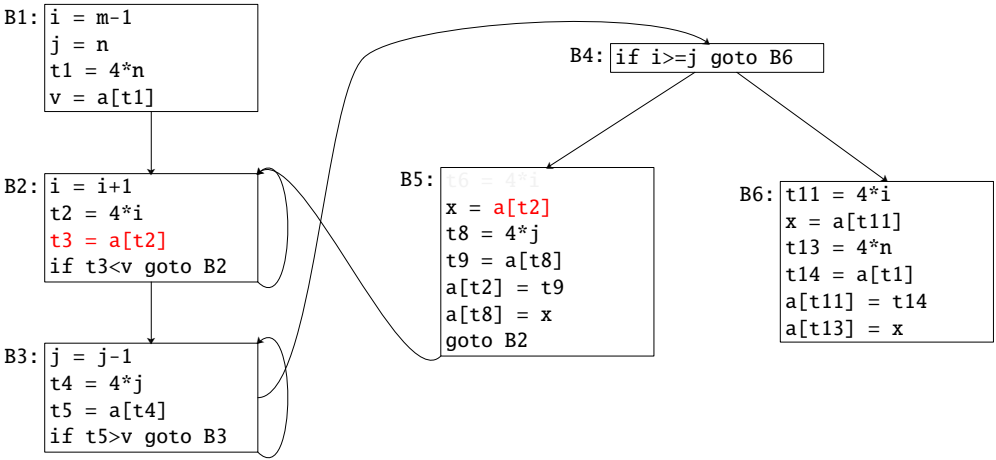
Global Common Subexpression Elimination



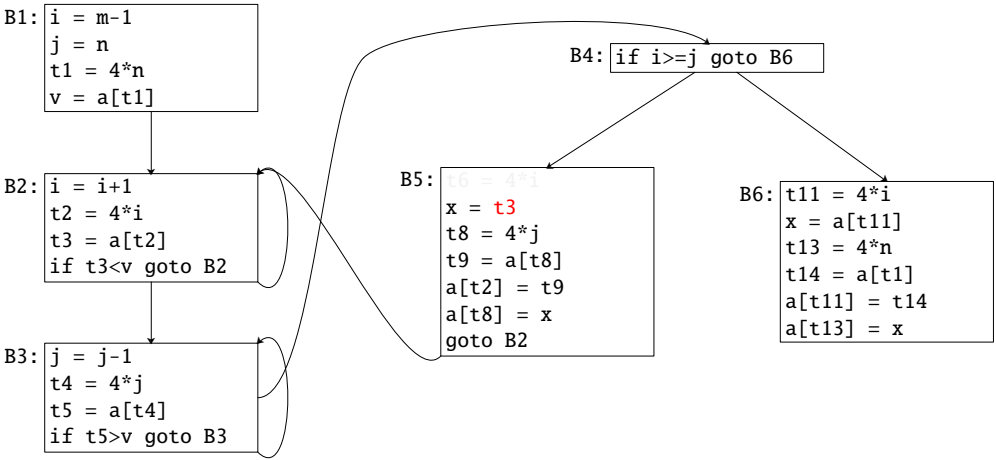
Global Common Subexpression Elimination



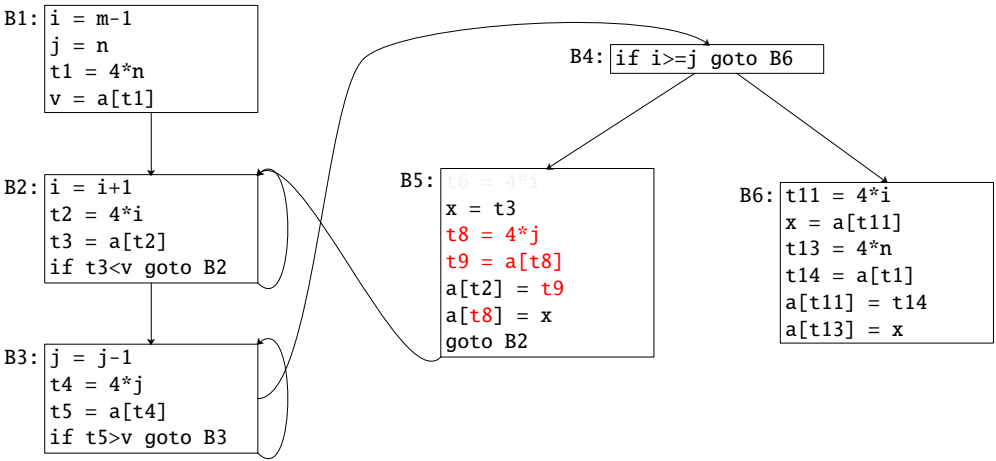
Global Common Subexpression Elimination



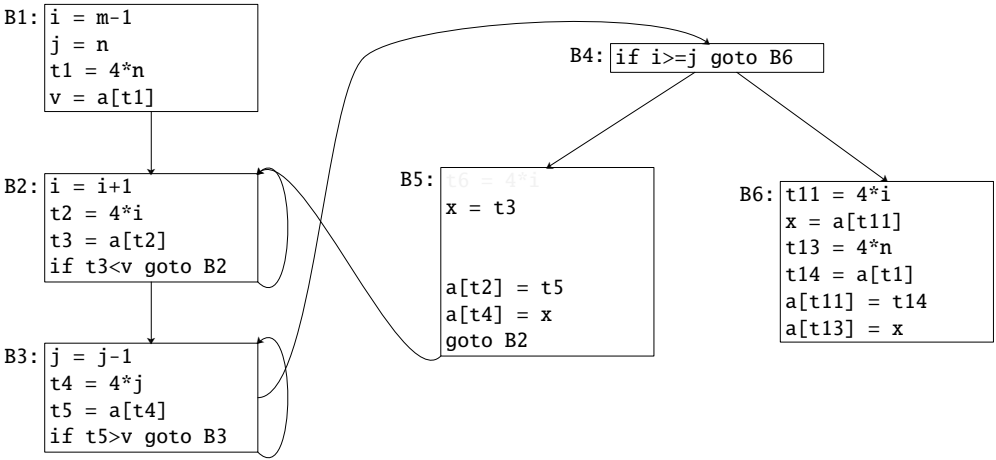
Global Common Subexpression Elimination



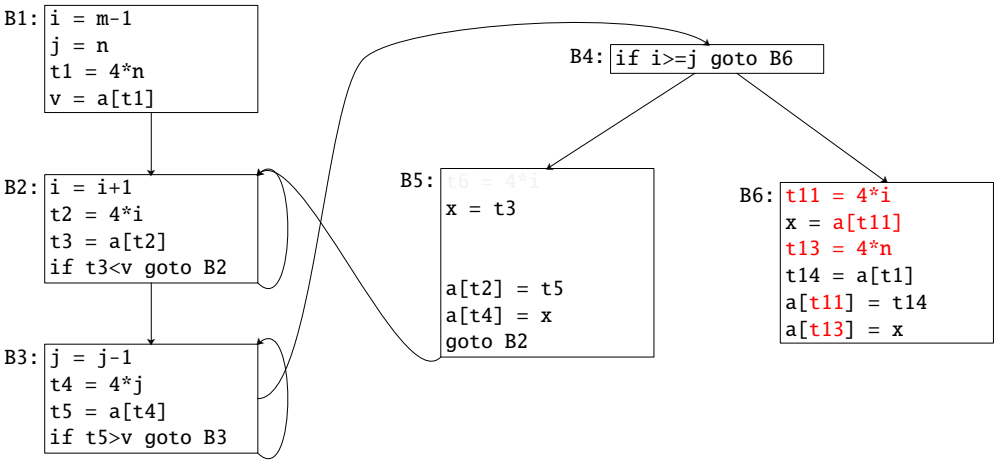
Global Common Subexpression Elimination



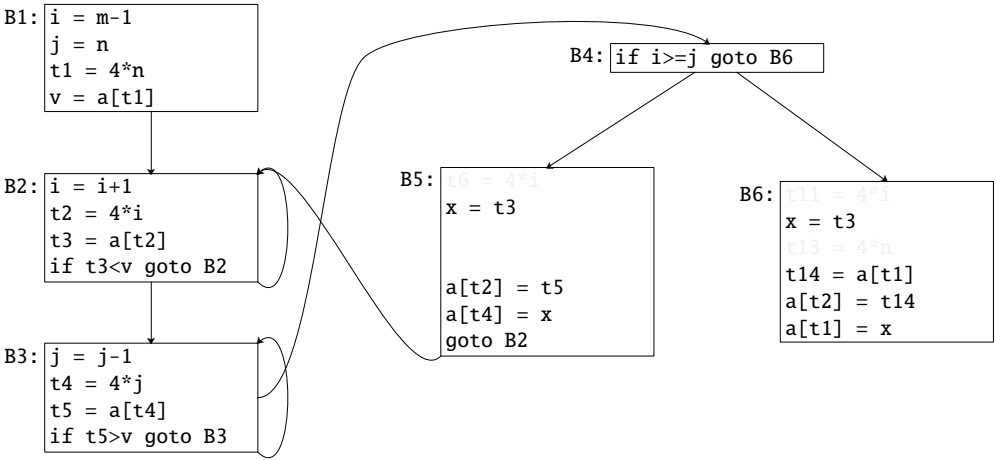
Global Common Subexpression Elimination



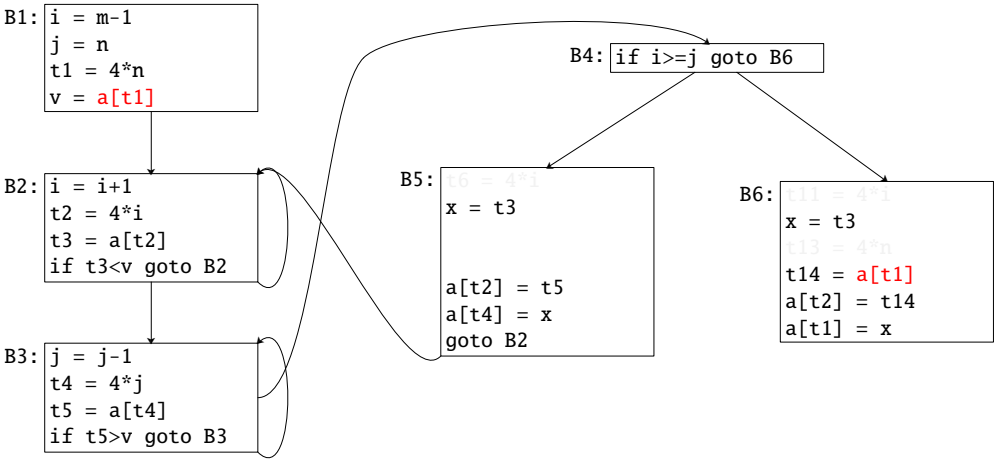
Global Common Subexpression Elimination



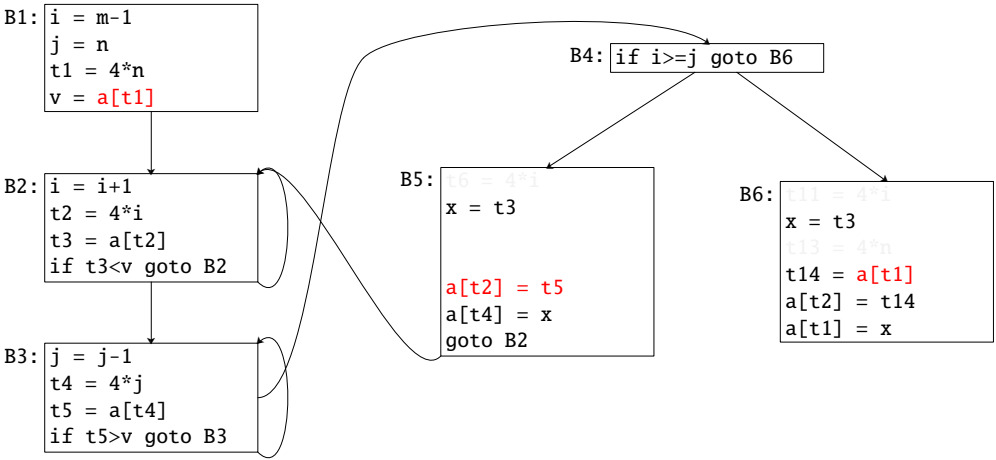
Global Common Subexpression Elimination



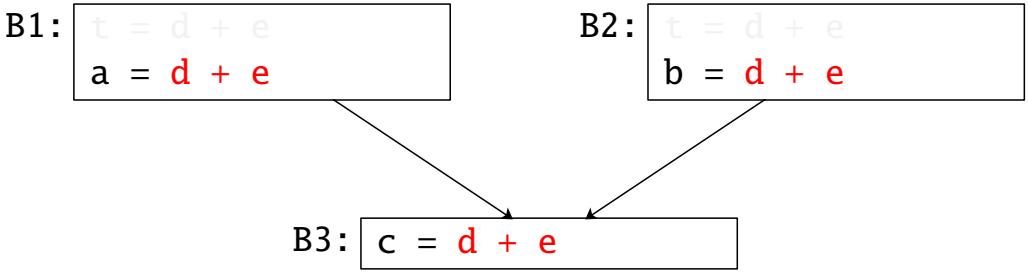
Global Common Subexpression Elimination



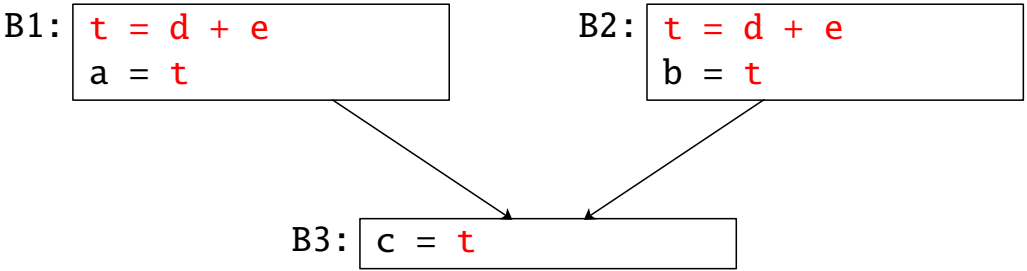
Global Common Subexpression Elimination



Copy Propagation



Copy Propagation



Code Motion

```
while (i <= limit-2) /*not changing limit*/
```

becomes

```
t = limit-2;
while (i <= t) /*not changing limit or t*/
```

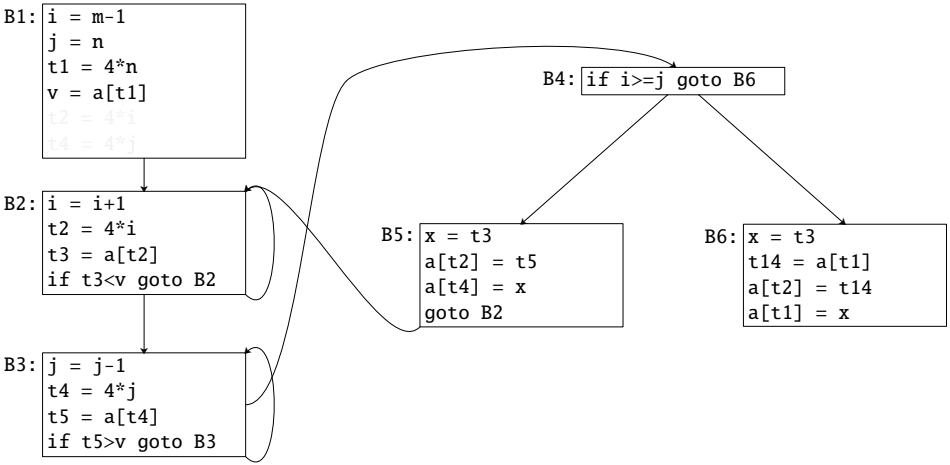
Code Motion

```
while (i <= limit-2) /*not changing limit*/
```

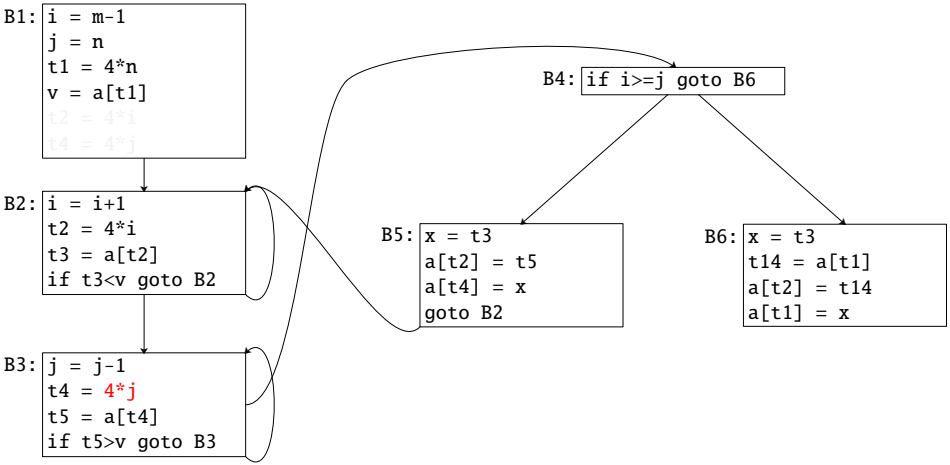
becomes

```
t = limit-2;
while (i <= t) /*not changing limit or t*/
```

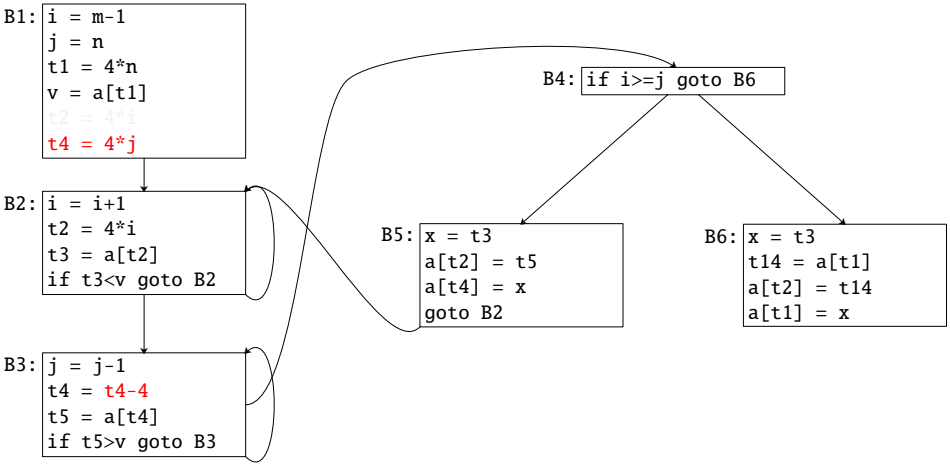
Induction Invariants and Reduction in Strength



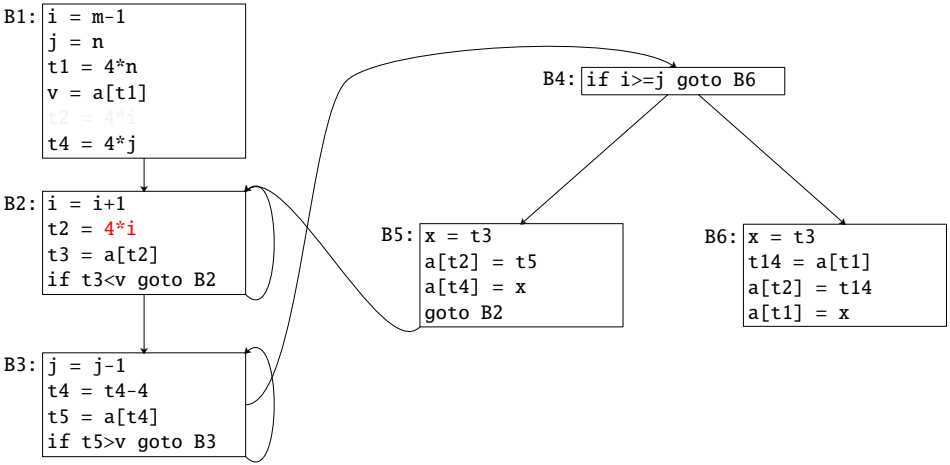
Induction Invariants and Reduction in Strength



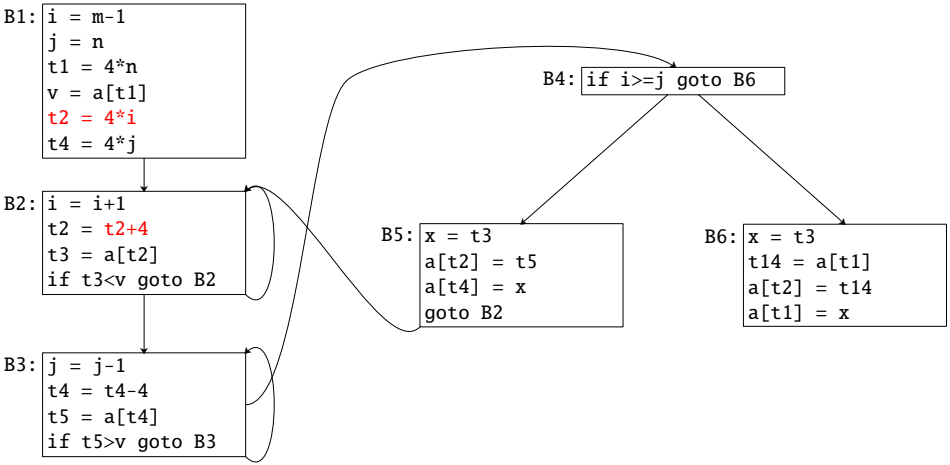
Induction Invariants and Reduction in Strength



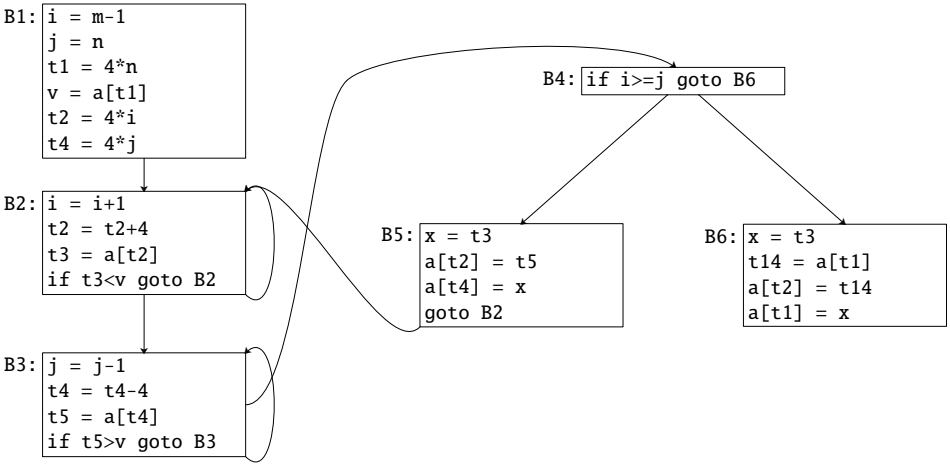
Induction Invariants and Reduction in Strength



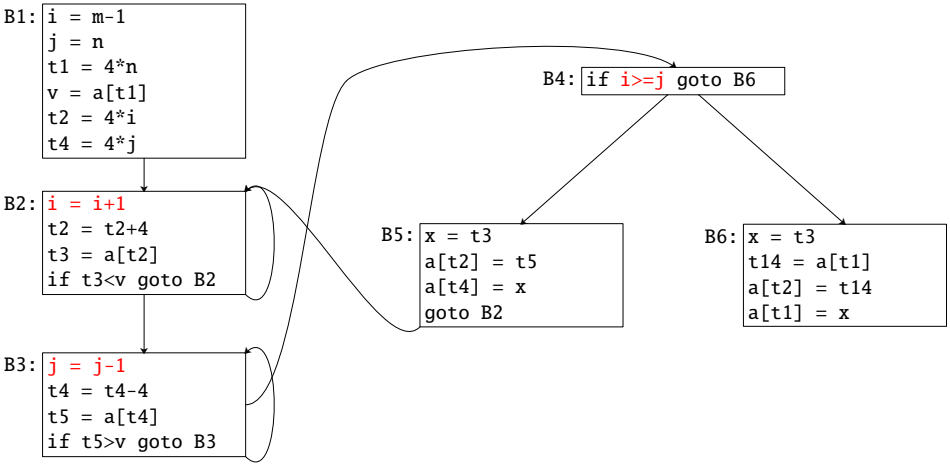
Induction Invariants and Reduction in Strength



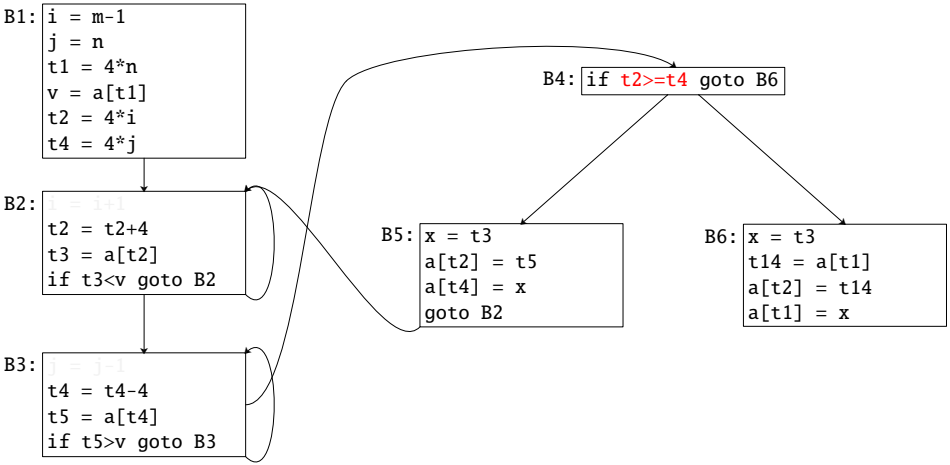
Induction Invariants and Reduction in Strength

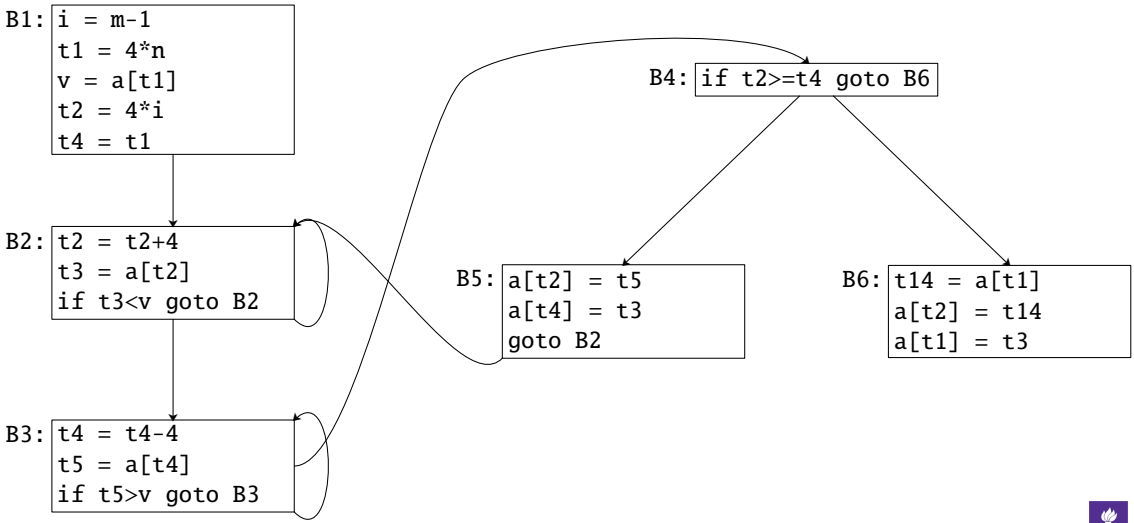


Induction Invariants and Reduction in Strength



Induction Invariants and Reduction in Strength




B1:
i = m-1
t1 = 4*n
v = a[t1]
t2 = 4*i
t4 = t1
B2:
t2 = t2+4
t3 = a[t2]
if t3<v goto B2
B3:
t4 = t4-4
t5 = a[t4]
if t5>v goto B3
B4:
if t2>=t4 goto B6
B5:
a[t2] = t5
a[t4] = t3
goto B2
B6:
t14 = a[t1]
a[t2] = t14
a[t1] = t3

Summary

Redundancy but preserve semantics!

- Global Common Subexpressions.
- Copy Propagation.
- Dead-code Elimination.
- Code Motion.
- Induction Variables/Strength Reduction.

Summary

Redundancy but preserve semantics!

- Global Common Subexpressions.
- Copy Propagation.
- Dead-code Elimination.
- Code Motion.
- Induction Variables/Strength Reduction.

Summary

Redundancy but preserve semantics!

- Global Common Subexpressions.
- Copy Propagation.
- Dead-code Elimination.
- Code Motion.
- Induction Variables/Strength Reduction.

Summary

Redundancy but preserve semantics!

- Global Common Subexpressions.
- Copy Propagation.
- Dead-code Elimination.
- Code Motion.
- Induction Variables/Strength Reduction.

Summary

Redundancy but preserve semantics!

- Global Common Subexpressions.
- Copy Propagation.
- Dead-code Elimination.
- Code Motion.
- Induction Variables/Strength Reduction.

Summary

Redundancy but preserve semantics!

- Global Common Subexpressions.
- Copy Propagation.
- Dead-code Elimination.
- Code Motion.
- Induction Variables/Strength Reduction.

Summary

Redundancy but preserve semantics!

- Global Common Subexpressions.
- Copy Propagation.
- Dead-code Elimination.
- Code Motion.
- Induction Variables/Strength Reduction.

Questions?

krisrose@cs.nyu.edu

- HACS will recover this week, and pr2 will be restarted.
- Because of GSAS conflict we swap final and pr3:
  - Final is May 12.
  - pr3 will be due week of May 19.

Questions?

krisrose@cs.nyu.edu

- HACS will recover this week, and pr2 will be restarted.
- Because of GSAS conflict we swap final and pr3:
  - Final is May 12.
  - pr3 will be due week of May 19.