CS3005D Compiler Design

Winter 2024

Lecture #30

Code Optimization, CFG

Saleena N
CSED NIT Calicut

March 2024

CSE: Identifying Redundancies

```
t1=i*4
t2=a[t1]
t3=i+1
i=t3
t4=i*4
t5=b[t4]
t6=t2+t5
x=t6
```

Is the second occurrence of `i*4`, redundant?

Exercise

Write the 3-address code generated for:

```
if(i<10)
   x=a[i]
else
   x=b[i]
sum=x
```

Generated code:

```
    if(i<10) goto L1
    goto L2
L1:t1=i*4
    t2=a[t1]
    x=t2
    goto L3
L2:t3=i*4
    t4=b[t3]
    x=t4
L3:sum=x
```

Is the second occurrence of `i*4`, redundant?

Identifying redundancies

```
    if(i<10) goto L1
    goto L2
L1:t1=i*4
    t2=a[t1]
    x=t2
    goto L3
L2:t3=i*4
    t4=b[t3]
    x=t4
L3:sum=x
```

Depending on the truth value of `i<10`, control flows to either L1 or L2.

`i*4` is evaluated only once.

The second occurrence of `i*4` is not redundant

```
    if(i<10) goto L1
    goto L2
L1:t1=i*4
    t2=a[t1]
    x=t2
    goto L3
L2:t3=i*4
    t4=b[t3]
    x=t4
L3:t5=i*4
    c[t5]=x
```

Any redundant computations?

```
    if(i<10) goto L1
    goto L2
L1:t1=i*4
    t2=a[t1]
    x=t2
    goto L3
L2:t3=i*4
    t4=b[t3]
    x=t4
L3:t5=i*4
    c[t5]=x
```

The last occurrence of `i*4` is redundant.

`i*4` is evaluated in both the branches of the conditional.

from each branch, control reaches L3.

How to identify such redundancies? How to eliminate?

```
    if(i<10) goto L1
    goto L2
L1:t1=i*4
    t2=a[t1]
    x=t2
    goto L3
L2:t3=i*4
    t4=b[t3]
    x=t4
L3:t5=i*4
    c[t5]=x
```

Identifying the redundancy

- requires information regarding the expressions that are computed along the control flow paths reaching `L3`.
- The information is obtained by doing a *data-flow analysis* of the program.

Data-flow Analysis

Derives information regarding the flow of data along program execution paths.

> *Available Expression Analysis* - computes the set of expressions available at every program point. Global CSE requires this information.

Analysis done in a Control Flow Graph representation of the program.

Control Flow Graph (CFG)

A graph representation of intermediate code.

- each Basic block is a node in the graph
- edge from block B_i to block B_j, if control flows from B_i to B_j

Basic Blocks

A maximal sequence of consecutive 3-address instructions such that

- flow of control can enter the basic block only through the first instruction in the block
- control leaves the block only after executing the last instruction in the block

Basic Blocks

Identify the basic blocks:

```
L1:t1=i*4
   t2=a[t1]
   x=t2
   goto L3
L2:t3=i*4
   t4=b[t3]
   x=t4
L3:sum=x
```

Basic Blocks

$$t1 = i * 4$$
$$t2 = a[t1]$$
$$x = t2$$
$$goto\ L3$$

$$t3 = i * 4$$
$$t4 = b[t3]$$
$$x = t4$$

$$sum = x$$

Basic Block: Leader

The first statement in a basic block is known as the *leader* of the Basic Block. A *leader* can be:

- the first instruction in the code
- an instruction that is the target of a conditional or unconditional jump
- instruction that follows a conditional or unconditional jump

Basic Block: Leader

Identify the leaders:

```
    if(i<10) goto L1
    goto L2
L1:t1=i*4
    t2=a[t1]
    x=t2
    goto L3
L2:t3=i*4
    t4=b[t3]
    x=t4
L3:sum=x
```

Basic Block: Leaders

```
    if(i<10) goto L1
    goto L2
L1:t1=i*4
    t2=a[t1]
    x=t2
    goto L3
L2:t3=i*4
    t4=b[t3]
    x=t4
L3:sum=x
```

Basic Blocks

$$if(i < 10)\ goto\ L1$$

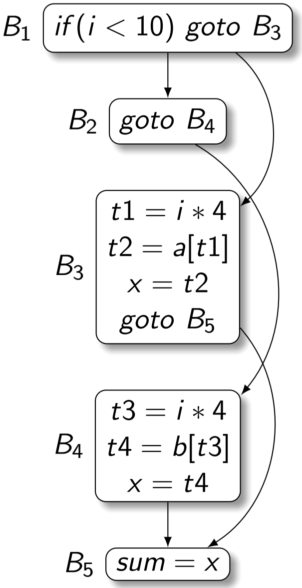
$$goto\ L2$$

$$\begin{gathered} t1 = i * 4 \\ t2 = a[t1] \\ x = t2 \\ goto\ L3 \end{gathered}$$

$$\begin{gathered} t3 = i * 4 \\ t4 = b[t3] \\ x = t4 \end{gathered}$$

$$sum = x$$

Control Flow Graph



References

References:

- Aho A.V., Lam M.S., Sethi R., and Ullman J.D. Compilers: Principles, Techniques, and Tools (ALSU). Pearson Education, 2007.

Further reading:

- ALSU Section 8.4, Chapter 9