The University of North Carolina at Chapel Hill

COMP 144 Programming Language Concepts

Spring 2002

# Lecture 36: Global Optimization, Loop Improvement and Profiling

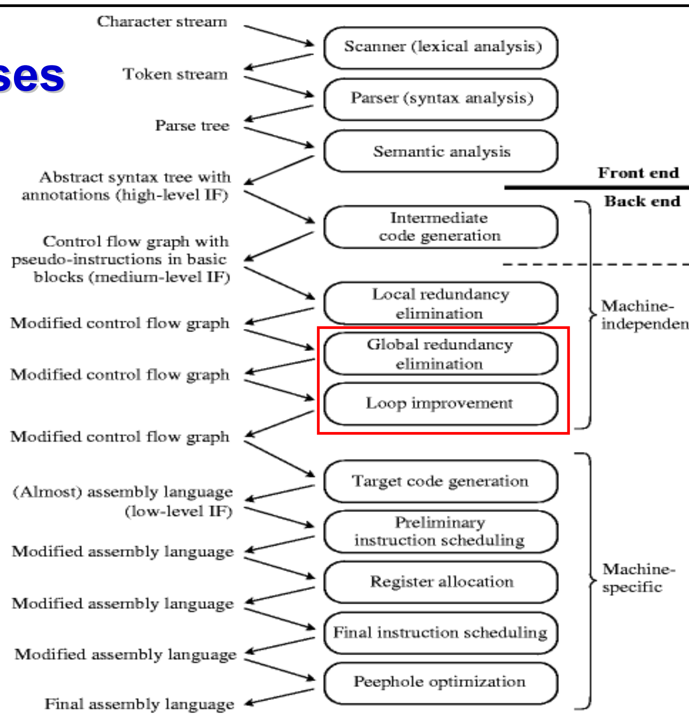
Felix Hernandez-Campos

April 22



## Phases

- We will discuss very briefly global optimization techniques and loop improvement



# Global Optimization

- *Local optimization* focuses on eliminating redundancies within basic block
- ***Global optimization*** focuses on eliminating redundancies across the boundaries between basic blocks
  - Static Single Assignment (SSA) form
  - Global value numbering



# Redundancy Elimination in Basic Blocks

- We will consider the example on the right
- It computes the binomial coefficients

$$\binom{n}{m}$$

for $0 \leq m \leq n$

- It is based on

$$\binom{n}{m} = \binom{n}{n-m}$$

```
combinations (int n, int *A) {
    int i, t;
    A[0] = 1;
    A[n] = 1;
    t = 1;
    for (i = 1; i <= n/2; i++) {
        t = (t * (n+1-i)) / i;
        A[i] = t;
        A[n-i] = t;
    }
}
```

## Naïve Control Flow Graph for the `combinations` subroutine

```
Block 1:
    sp := sp − 8
    v1 := r4        −− n
    n := v1
    v2 := r5        −− A
    A := v2

    v3 := A
    v4 := 1
    *v3 := v4
    v5 := A
    v6 := n
    v7 := 4
    v8 := v6 x v7
    v9 := v5 + v8
    v10 := 1
    *v9 := v10
    v11 := 1
    t := v11
    v12 := 1
    i := v12
    goto Block 3
```

```
Block 2:
    v13 := t
    v14 := n
    v15 := 1
    v16 := v14 + v15
    v17 := i
    v18 := v16 − v17
    v19 := v13 x v18
    v20 := i
    v21 := v19 div v20
    t := v21
    v22 := A
    v23 := i
    v24 := 4
    v25 := v23 x v24
    v26 := v22 + v25
    v27 := t
    *v26 := v27
    v28 := A
    v29 := n
    v30 := i
    v31 := v29 − v30
    v32 := 4
    v33 := v31 x v32
    v34 := v28 + v33
    v35 := t
    *v34 := v35
    v36 := i
    v37 := 1
    v38 := v36 + v37
    i := v38
    goto Block 3
```

```
Block 3:
    v39 := i
    v40 := n
    v41 := 2
    v42 := v40 div v41
    v43 := v39 ≤ v42
    if v43 goto Block 2
    else goto Block 4
```

```
Block 4:
    sp := sp + 8
    goto *ra
```

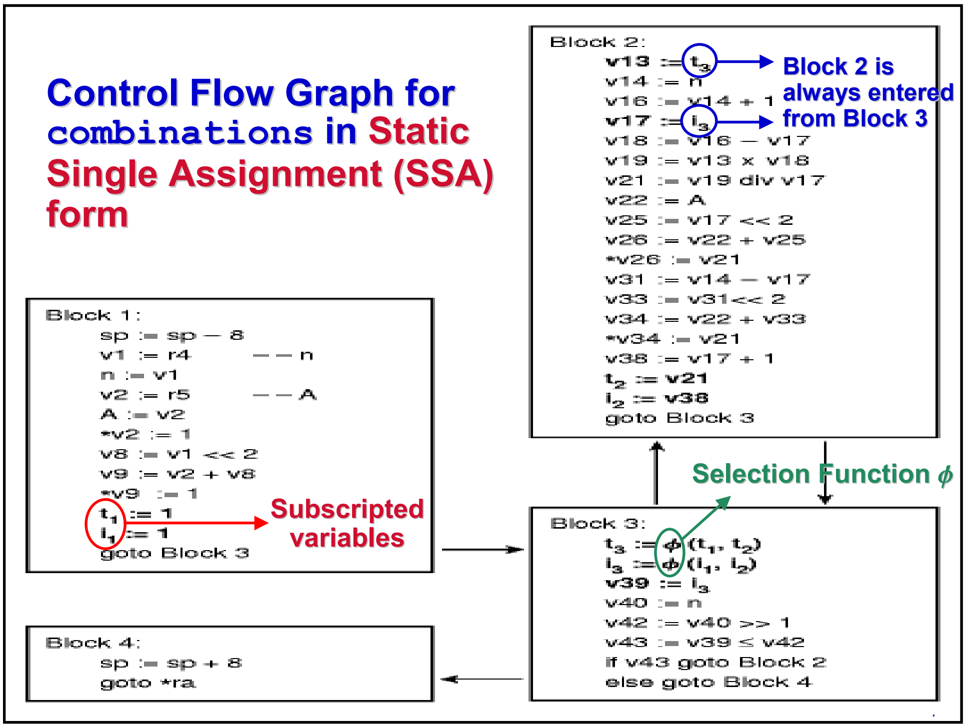
## Control Flow Graph for `combinations` after *local redundancy elimination* and *strength reduction*

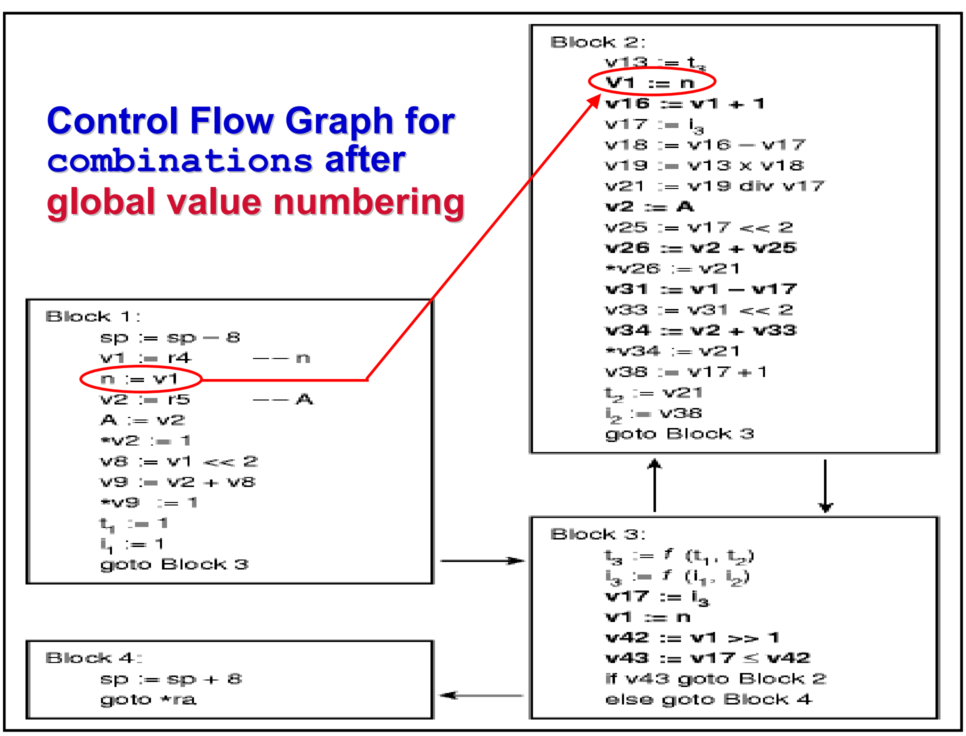
```
Block 1:
    sp := sp − 8
    v1 := r4        −− n
    n := v1
    v2 := r5        −− A
    A := v2
    *v2 := 1
    v8 := v1 << 2
    v9 := v2 + v8
    *v9  := 1
    t := 1
    i := 1
    goto Block 3
```

```
Block 2:
    v13 := t
    v14 := n
    v16 := v14 + 1
    v17 := i
    v18 := v16 − v17
    v19 := v13 x v18
    v21 := v19 div v17
    v22 := A
    v25 := v17 << 2
    v26 := v22 + v25
    *v26 := v21
    v31 := v14 − v17
    v33 := v31 << 2
    v34 := v22 + v33
    *v34 := v21
    v38 := v17 + 1
    t := v21
    i := v38
    goto Block 3
```

```
Block 3:
    v39 := i
    v40 := n
    v42 := v40  >> 1
    v43 := v39 ≤ v42
    if v43 goto Block 2
    else goto Block 4
```

```
Block 4:
    sp := sp + 8
    goto *ra
```


Control Flow Graph for combinations in Static Single Assignment (SSA) form
Block 1:
sp := sp − 8
v1 := r4 −− n
n := v1
v2 := r5 −− A
A := v2
*v2 := 1
v8 := v1 << 2
v9 := v2 + v8
*v9 := 1
t1 := 1
i1 := 1
goto Block 3
Subscripted variables
Block 2:
v13 := t3
v14 := n
v16 := v14 + 1
v17 := i3
v18 := v16 − v17
v19 := v13 x v18
v21 := v19 div v17
v22 := A
v25 := v17 << 2
v26 := v22 + v25
*v26 := v21
v31 := v14 − v17
v33 := v31 << 2
v34 := v22 + v33
*v34 := v21
v38 := v17 + 1
t2 := v21
i2 := v38
goto Block 3
Block 2 is always entered from Block 3
Selection Function ϕ
Block 3:
t3 := ϕ (t1, t2)
i3 := ϕ (i1, i2)
v39 := i3
v40 := n
v42 := v40 >> 1
v43 := v39 ≤ v42
if v43 goto Block 2
else goto Block 4
Block 4:
sp := sp + 8
goto *ra


Control Flow Graph for combinations after global value numbering
Block 1:
sp := sp − 8
v1 := r4 −− n
n := v1
v2 := r5 −− A
A := v2
*v2 := 1
v8 := v1 << 2
v9 := v2 + v8
*v9 := 1
t1 := 1
i1 := 1
goto Block 3
Block 2:
v13 := t3
V1 := n
v16 := v1 + 1
v17 := i3
v18 := v16 − v17
v19 := v13 x v18
v21 := v19 div v17
v2 := A
v25 := v17 << 2
v26 := v2 + v25
*v26 := v21
v31 := v1 − v17
v33 := v31 << 2
v34 := v2 + v33
*v34 := v21
v38 := v17 + 1
t2 := v21
i2 := v38
goto Block 3
Block 3:
t3 := f (t1, t2)
i3 := f (i1, i2)
v17 := i3
v1 := n
v42 := v1 >> 1
v43 := v17 ≤ v42
if v43 goto Block 2
else goto Block 4
Block 4:
sp := sp + 8
goto *ra

## Global Optimization

- After global value numbering, data flow analysis is used to determine redundancies across block boundaries
- The main two tasks are *common subexpression elimination* and *live variable analysis*

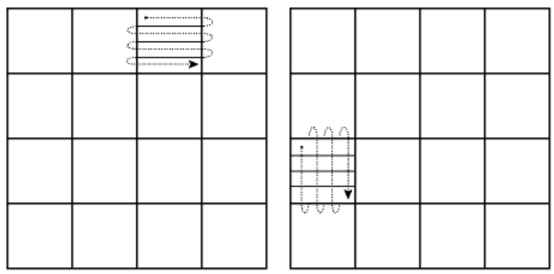


## Loop Improvement

- Programs spend most of their time in loops
- Consequently, it is particularly important to generate good code for loops
- Common techniques:
  - Relocate *loop invariant* expressions outside the loop
  - Reduce the amount of time spent maintaining *induction variables* (*i.e.*, loop indexes)
- Advanced techniques:
  - *Loop unrolling* for improved instruction scheduling
  - *Loop reordering* for optimized cache access patterns and concurrent execution

## Memory Latency Is Critical

- There is a huge gap between memory latency and processor latency (including cache latency)
- It is critical to optimize the use of the cache to reduce as much as possible the number of memory accesses





## Optimization

- Programming language performance varies greatly
  - The Great Computer Language Shootout
    - » http://www.bagley.org/~doug/shootout/
- Programming language constructs within the same language may varies greatly in their performance
  - *Python Patterns - An Optimization Anecdote*
    - » http://www.python.org/doc/essays/list2str.html
  - `Time()`
    - » http://www.python.org/doc/current/lib/module-time.html#l2h1382
  - *Python Performance Tips*
    - » http://manatee.mojam.com/~skip/python/fastpython.html

# Reading Assignment

- Read Scott
  - Sect. 13.4.1
  - Rest of chapter 13 (only general ideas, no details)